INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Uma Ferramenta para Execução de Processos de Negócio em OutSystems

Hélio Filipe Coelho de Almeida

Dissertação para obtenção do grau de Mestre em

Engenharia Informática e Computadores

Júri

Presidente: Pedro Manuel Moreira Vaz Antunes de Sousa

Orientador: Miguel Leitão Bignolas Mira da Silva

Vogais: Diogo Manuel Ribeiro Ferreira

Julho de 2007

Agradecimentos

Gostaria de expressar o meu agradecimento a todos aqueles que contribuíram para o trabalho aqui apresentado, sem os quais este não seria possível.

Em primeiro lugar gostaria de agradecer ao professor Miguel Mira da Silva pelo acompanhamento que fez e por todo o apoio que prestou. O seu largo conhecimento desta área levou a que as suas sugestões fossem bastante úteis para o desenvolvimento de um trabalho correcto e direccionado.

Agradeço igualmente a contribuição do professor Diogo Ferreira que ajudou bastante quer na investigação, quer na sustentação teórica e cujos conselhos ajudaram em momentos chave deste trabalho. Gostaria também de referir a ajuda preciosa do professor Artur Caetano durante a investigação inicial deste trabalho.

No que respeita à OutSystems, expresso um agradecimento especial ao Hugo Lourenço e ao Lúcio Ferrão cujo acompanhamento que fizeram deste trabalho teve um valor incalculável. O seu conhecimento profissional e sobre a OutSystems é imenso. Trabalhámos em conjunto para que a solução desenvolvida fosse a melhor possível. Agradeço também ao Rodrigo Coutinho e ao Carlos Alves cuja supervisão e intervenções ajudaram a manter o resultado deste trabalho alinhado com as necessidades que o motivaram. Gostaria de agradecer aos profissionais do "*Solutions Delivery*" da OutSystems que por várias vezes partilharam a sua experiência profissional. A toda a equipa da OutSystems um muito obrigado, por toda a ajuda que prestaram, pela forma como me receberam, e pela boa disposição que sempre apresentaram.

Finalmente devo um agradecimento muito especial ao meu colega Fernando Graça, também finalista que paralelamente ao meu trabalho desenvolveu a solução de modelação de processos de negócio na plataforma OutSystems. O seu empenho, profissionalismo e facilidade com que encara as dificuldades mostraram-se importantíssimos para o trabalho desenvolvido.

A todos os que referi e a todos os que não tenha referido mas que tenham ajudado directa ou indirectamente no desenvolvimento deste trabalho, deixo um profundo *Muito Obrigado*.

Abstract

The current interest in business processes comes from the necessity to organize operations and to consolidate organizations allowing them to become more adaptable. The Business Process Management, the disciplines that study this subject, is supported by a set of tools named BPMS's *(Business Process Management Suites)*, that address the business processes of an organization in order to supply the needed agility. With the recent developments in IT a new type of tools based on agile methodologies appeared. This new tools answer the necessities of agility and adaptability of the organizations. OutSystems is a company that developed one of these new tools that despite not having the objective of competing with the BPMS's, had the necessity of including business processes in their tool in order to be lined up with the its customers necessities. The present document describes the work done to provide the OutSystems platform with capacity for business process execution. This way the OutSystems platform becomes one of the few tools with support for business processes that does not need written programming in order to develop processes or applications.

Keywords

Business Process, Execution, Platform, Process Definition, Process Instance, *Token*, Execution Engine.

Resumo

O interesse actual nos processos de negócio advém da necessidade de organizar operações e consolidar organizações tornando-as mais adaptáveis. A gestão de processos de negócio, disciplina que aborda esta área, apoia-se num conjunto de ferramentas designadas por BPMS's *(Business Process Management Suites),* que permitem abordar os processos de negócio de uma organização de modo a fornecer a agilidade pretendida. Com os avanços recentes nas TI surgiu um novo tipo de ferramentas baseadas em metodologias ágeis que respondem às necessidades de agilidade e adaptabilidade das organizações. A OutSystems é uma empresa que desenvolveu uma destas novas ferramentas que embora não tivesse como objectivo a competição com as BPMS's teve a necessidade de incluir processos de negócio, de modo a ficar alinhada com as necessidades dos seus clientes. O documento presente relata o trabalho desenvolvido para se dotar a plataforma OutSystems com capacidade de execução de processos de negócio. Desta forma a plataforma OutSystems passa a ser uma das poucas ferramentas com suporte para processos de negócio, que não necessita de programação escrita para o desenvolvimento de processos e aplicações.

Palavras-chave

Processo de Negócio, Execução, Plataforma, Definição de Processo, Instância de Processo, *Token*, Motor de Execução.

Índice

Agradecimentos I

Abstract II

Resumo II

1. Introdução 1

1.1 Plataforma OutSystems 2

2. Contextualização 7

2.1 Gestão de Processos de Negócio 7

2.2 Execução de Processos de Negócio 12

2.3 Ferramentas Actuais de Execução de Processos de Negócio 18

3. Problema 25

3.1 Ambiente de Negócio 25

3.2 OutSystems 26

3.3 Motor de Execução Externo 27

4. Proposta 30

4.4 Requisitos 30

4.1 Abordagem 33

4.2 Arquitectura 34

5. Implementação 40

5.1 Metodologia 40

5.2 Gerador de Código 40

5.3 Motor de Execução 45

5.4 Modelo de Dados 53

5.5 Contexto 55

6. Resultados 57

6.1 Caso de Estudo 57

6.2 Geração de Código 61

6.3 Execução 65

7. Conclusão 69

7.1 Trabalho Realizado 70

7.2 Trabalho Futuro 71

8. Referências 72

Anexos

1. BPEL
2. Aplicação de Visualização de Resultados
3. Output da Geração do Processo Exemplo
4. Glossário

Figuras

Figura 1 : Componentes da Plataforma OutSystems 4
Figura 2 : Plataforma – *Upload*, Compilação e Instalação 5
Figura 3 : Processo de publicação e execução 6
Figura 4: Processo de ciclo de vida dos processos 9
Figura 5 : Definição de Petri Nets 14
Figura 6 : Petri Nets Implementation 15
Figura 7 : Petri Nets Implementation 15
Figura 8 : Separação dos sistemas de gestão de *workflow* segundo a norma WAPI 16
Figura 9 : *IBM WebSphere Suite Architecture* 19
Figura 10 : *BEA AquaLogic BPM Suite Architecture* 20
Figura 11 : *Savvion BusinessManager Suite Architecture* 22
Figura 12 : Estrutura de plataforma com um componente de execução externo 29
Figura 13 : Elementos da linguagem de modelação 31
Figura 14 : Funções da Process API 32
Figura 15 : Adaptações necessárias à plataforma 34
Figura 16 : Inputs e respectivos outputs produzidos durante a geração de código 35
Figura 17 : Arquitectura do Motor de Execução 36
Figura 18 : Estrutura da Base de Dados 38
Figura 19 : Arquitectura Global 39
Figura 20 : Novos elementos da OML 41
Figura 21 : Diagrama de classes 42
Figura 22 : Exemplo da área de influência do nó Join 45
Figura 23 : Definições, Instâncias e *Tokens* 46
Figura 24 : Processamento de um pedido 48
Figura 25 : Execução do *Token* 49
Figura 26 : Diagrama de classes do Motor de Execução 50
Figura 27 : Modelo de dados 53
Figura 28 : Execução de um sub-processo 55
Figura 29 : Processo Material Request 58
Figura 30 : Ecrã de submissão de pedidos 59

Figura 31 : Ecrã de aprovação por parte do subdirector 60
Figura 32 : Ecrã de aprovação por parte do Director 60
Figura 33 : Detalhes da definição do processo *MaterialRequest* 61
Figura 34 : Exemplo de nós da definição do processo *MaterialRequest* 61
Figura 35 : Exemplo de ligações entre nós da definição do processo *MaterialRequest* 62
Figura 36 : Tabela de tipos preenchida 62
Figura 37 : Construtor da classe da definição gerada 63
Figura 38 : Métodos associados com a lógica das tarefas 64
Figura 39 : Métodos para criação de instâncias de processo 64
Figura 40 : Criação de um pedido de material 65
Figura 41 : Instância de processo criada 66
Figura 42 : *Token* à espera do evento de aprovação do subdirector 66
Figura 43 : Pedido à espera de aprovação do subdirector 66
Figura 44 : *Token* à espera do evento de aprovação do director 67
Figura 45 : Pedido à espera de aprovação do director 67
Figura 46 : Lista de *Tokens* após o final da execução do processo 68
Figura 47 : Instância do processo após o final da execução do processo 68

Tabelas

Tabela 1 : Tradução Modelação – Execução 11
Tabela 2 : Comparação das ferramentas de BPM 24

1. Introdução

As incessantes alterações das condições de mercado que se verificam actualmente levaram a uma adaptação contínua por parte das organizações. A capacidade de resposta, fortemente ligada à flexibilidade das organizações, é o factor que dita a sobrevivência e bem-estar no mercado.

A estruturação das organizações tendo em conta os seus processos de negócio tornou-se uma abordagem recorrente por permitir melhorar cada vez mais os processos internos às organizações, cortando custos e maximizando a eficácia. Desta forma aumentou o interesse na área da gestão de processos de negócio.

O principal modo pelo qual os processos oferecem a flexibilidade desejada pelas organizações é a possibilidade de executar processos, tornando imediatamente reais as alterações desenvolvidas durante a fase de modelação. Isto permite aos elementos das organizações alterarem e melhorarem os seus processos sempre que necessário e ver as suas alterações repercutidas na organização, podendo voltar a ser alteradas caso necessário. Em ultima análise a fase de execução é essencial para a flexibilidade necessária para as organizações se manterem de acordo com as inconstantes condições de mercado.

Com o aumento do interesse na gestão de processos de negócio, surgiram várias novas tecnologias que pretendem abordar a execução de processos de negócio. Surgiram novas linguagens denominadas de linguagens de execução de processos de negócio, que permitem detalhar um processo de negócio para que este possa ser executado sobre esta linguagem, surgiram tecnologias úteis para o desenvolvimento de componentes de execução de processos e surgiu um novo conjunto de ferramentas as BPMS's que abordam a execução de processos e todas as outras fases que compões a gestão de processos de negócio.

As BPMS's oferecem um novo método para gerir os processos, reunindo numa única ferramenta todas as actividades inerentes à gestão dos processos de negócio numa organização. A principal vantagem destas ferramentas prende-se com a facilidade de modelação e execução dos processos de negócio que se pretendem gerir e que era desconhecida até ao momento do seu aparecimento.

Com os recentes avanços nas tecnologias de informação têm começado a surgir, novas ferramentas com métodos de desenvolvimento ágeis, que oferecem uma flexibilidade extrema nas suas aplicações. Estas novas ferramentas permitem que as aplicações desenvolvidas sejam facilmente alteradas, sempre que necessário, para fazer face às alterações da organização. Embora baseadas em outras tecnologias, o resultado final destas ferramentas é de certa forma semelhante ao da gestão de processos de negócio, ou seja, conseguem dotar as organizações da flexibilidade necessária às condições de negócio actuais.

O facto de actualmente as organizações se encontrarem estruturadas de acordo com os seus processos de negócio faz com que apesar de bastante ágeis, estas ferramentas não estejam

alinhadas com as organizações. Embora as aplicações desenvolvidas sejam bastante ágeis, não permitem endereçar os processos de negócio que representam o modo como actualmente as organizações actuam, evoluem e em ultima análise são geridas.

A OutSystems é um exemplo de uma empresa que comercializa uma destas novas ferramentas baseadas em metodologias de desenvolvimento ágeis. Junto dos seus clientes, surgiu a necessidade de suporte de processos de negócio de modo a desenvolver aplicações mais orientadas ao negócio, havendo assim necessidade de dotar a plataforma com suporte para as actividades inerentes à gestão de processos de negócio, onde se enquadra a *Execução*.

No decorrer deste documento oferece-se uma breve introdução sobre a estrutura da plataforma OutSystems tal como se encontrava quando este trabalho foi iniciado. Segue-se uma introdução à disciplina da Gestão de Processos de Negócio, algumas das tecnologias actualmente usadas na execução de processos de negócio e apresentam-se ainda algumas BPMS's do mercado. Prossegue--se com uma descrição em mais detalhe do problema que levou a este trabalho. Finalmente apresenta-se a proposta ao problema, implementação e obviamente a avaliação dos resultados obtidos no que respeita a inclusão do suporte de execução de processos na plataforma.

1.1 Plataforma OutSystems

A Plataforma OutSystems [21] destina-se principalmente ao desenvolvimento de aplicações empresariais com uma estrutura *web-based*. Esta tem suporte para redes móveis e de *e-mail* e permite integração com os sistemas *legacy* normalmente existentes nas organizações actuais.

A principal diferença em relação a outras ferramentas semelhantes assenta na metodologia de desenvolvimento proposta e na flexibilidade apresentada.

Segue-se uma introdução à plataforma OutSystems onde será abordada a metodologia de desenvolvimento que esta promove, os seus componentes e finalmente uma visão mais detalhada do componente responsável pela execução das aplicações desenvolvidas.

1.1.1 Nova Metodologia

Com a sua plataforma a *OutSystems* sugere uma aproximação diferente para o controlo e organização de projectos baseada em metodologias ágeis [1], [18]. A *OutSystems Agile Methodology* [22] aborda a actual necessidade de rápido desenvolvimento e contínua mudança das aplicações desenvolvidas, permitindo a criação de aplicações que respeitem quer as necessidades tecnológicas, quer as necessidades de negócio das organizações. Esta metodologia surgiu da adaptação dos conceitos da *SCRUM Agile Methodology* [26], [28] às características da plataforma criada.

Os projectos que seguem a *OutSystems Agile Methodology* são compostos por uma sequência de iterações, denominadas de *Sprints,* no final das quais uma versão funcional limitada do sistema

está pronta. Estas iterações, com duração de uma ou duas semanas, são desta forma compostas por actividades de análise, desenvolvimento e teste, no final das quais uma ou mais funcionalidades do sistema final é terminada. No final de todas as iterações consegue-se um sistema com todas as funcionalidades disponíveis, que foram sendo testadas, adaptadas e aprovadas paralelamente com o seu desenvolvimento.

Apoiando-se na metodologia anterior a *OutSystems* promove uma aproximação *built-to-change*, [3] na qual, independentemente da fase do ciclo de vida das aplicações novas funcionalidades podem ser facilmente adicionadas, erros corrigidos e feedback analisado, com riscos reduzidos e sem graves consequências para o negócio.

Ao contrário do comum das ferramentas de desenvolvimento, esta plataforma aposta num estilo de programação visual *drag'n'drop* sendo possível a criação de aplicações sem ter de se escrever qualquer linha de código. Desde o desenho do modelo de dados, criação de Interfaces, definição de lógica de negócio ou instalação, tudo pode ser feito visualmente.

Esta abordagem permite diminuir o desalinhamento existente entre o negócio e as TI, pois torna as aplicações mais fáceis de compreender para os elementos do negócio e permite aos elementos das TI responder atempadamente às necessidades que lhes são apresentadas.

1.1.2 Estrutura da Plataforma

Esta plataforma é destinada ao desenvolvimento de aplicações para Internet/Intranet ou redes móveis e é composta pelos quatro componentes apresentados na Figura 1.

Estes pretendem endereçar as fases de desenvolvimento, integração de sistemas, execução e monitorização das aplicações criadas.

Segue-se uma descrição mais detalhada de cada um destes componentes:

Service Studio: Componente de desenvolvimento visual, destinado à criação, alteração e instalação das aplicações desenvolvidas. Todo o processo de desenvolvimento é realizado neste componente, desde o desenho e criação do modelo de dados, desenho de interfaces e criação da lógica de negócio. A instalação das aplicações é feita neste componente recorrendo ao processo denominado *1-Click-Publishing*, o qual verifica, guarda, efectua o *upload* no componente de execução, compila e instala a aplicação. Se todo este processo decorrer sem erros obtém-se uma aplicação completamente executável.

Integration Studio: Componente de integração. Neste componente é possível fazer a integração com diversos sistemas *legacy*, quer através de *wizards* disponibilizados ou recorrendo à programação tradicional, oferecendo assim flexibilidade para integrar com qualquer tipo de sistema. Uma vez publicados este adaptadores podem ser utilizados na componente de desenvolvimento como blocos visuais para permitir a interacção das aplicações com os sistemas existentes.

Hub Server: Componente central responsável pela execução. Este orquestra todas as compilações, instalações e qualquer actividade que decorra em tempo de execução. Todos os objectos desenvolvidos necessitam de ser aqui publicados para que possam ser usados nos vários componentes.

Service Center: Componente de monitorização e gestão das aplicações. Este componente permite coordenar todos os objectos necessários à execução, desde aplicações, serviços, adaptadores e quaisquer outros recursos.

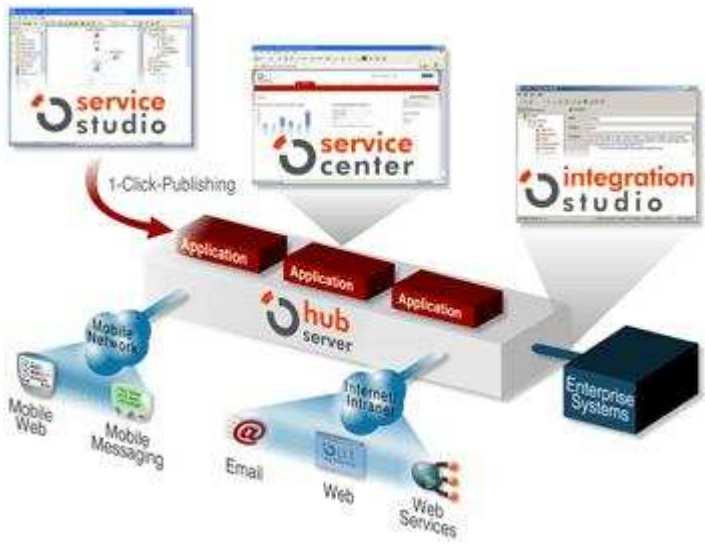


Figura 1 : Componentes da Plataforma OutSystems **[24]**

Com o conjunto dos componentes descritos anteriormente a plataforma OutSystems aborda praticamente todos os factores necessários ao desenvolvimento de aplicações empresariais.

1.1.3 HubServer

Para melhor se perceber as propostas e o trabalho realizado, segue-se uma descrição mais detalhada de algumas partes do componente de execução.

Como se pode ver pela Figura 2 após a ordem de publicação, proveniente do *Service Studio*, é enviado para o *HubServer* o ficheiro com a definição completa e detalhada do que foi desenvolvido visualmente no componente de desenvolvimento. Este ficheiro *oml* está estruturado de acordo com a linguagem interna *OutSystems Markup Language*.

Após o *upload* do ficheiro com a definição da aplicação este é processado e utilizado num gerador de código responsável por gerar o todo o código da aplicação que anteriormente tinha sido desenvolvida visualmente. São geradas as classes, as *queries* SQL, os *ecrans* e tudo o necessário à execução da aplicação.

Uma vez gerado o código da aplicação, este é compilado e os resultados desta compilação, bem como os da geração de código são instalados. A instalação corresponde a colocar os ficheiros criados nos locais respectivos para que possam ser acedidos durante a execução, a criar tabelas na base de dados ou reflectir alterações efectuadas, agendar serviços, produzir informação para a componente de monitorização e ainda a publicar a aplicação no sub-componente que ficará à escuta de novos pedidos à aplicação.

Qualquer erro que seja encontrado durante este processo é reportado ao utilizador, que ficará responsável por o corrigir e republicar a aplicação em mãos.

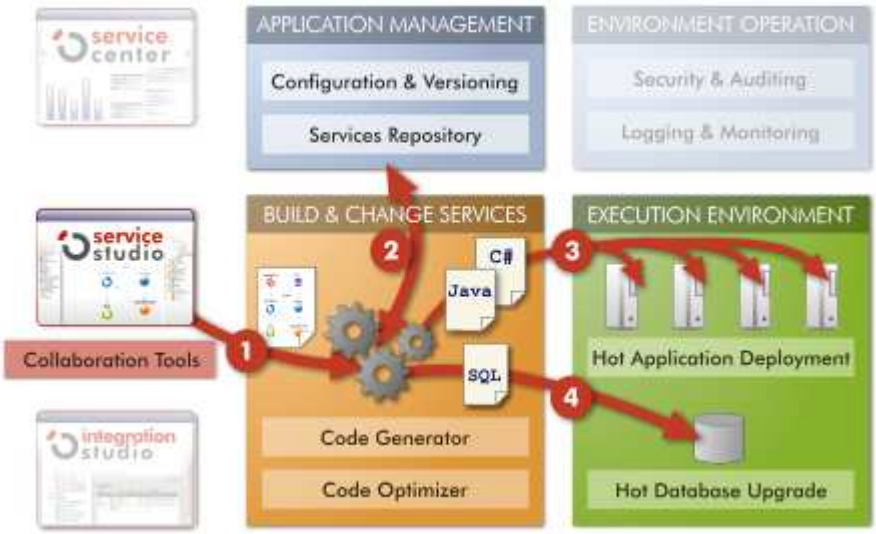


Figura 2 : Plataforma – *Upload*, Compilação e Instalação **[23]**

Após este processo a aplicação está pronta para executar. Qualquer pedido que lhe chegue vai utilizar os executáveis criados anteriormente que irão actuar quer sobre a base de dados, quer sobre outros ficheiros ou sistemas com que seja suposto a aplicação comunicar. A Figura 3 mostra sumariamente o processo de publicação nas setas a vermelho e os pedidos que surgem durante a execução nas setas a verde.

O *HubServer*, componente de execução, contém muito mais elementos que não foram relatados, no entanto a descrição anterior é necessária para melhor compreender o trabalho apresentado.

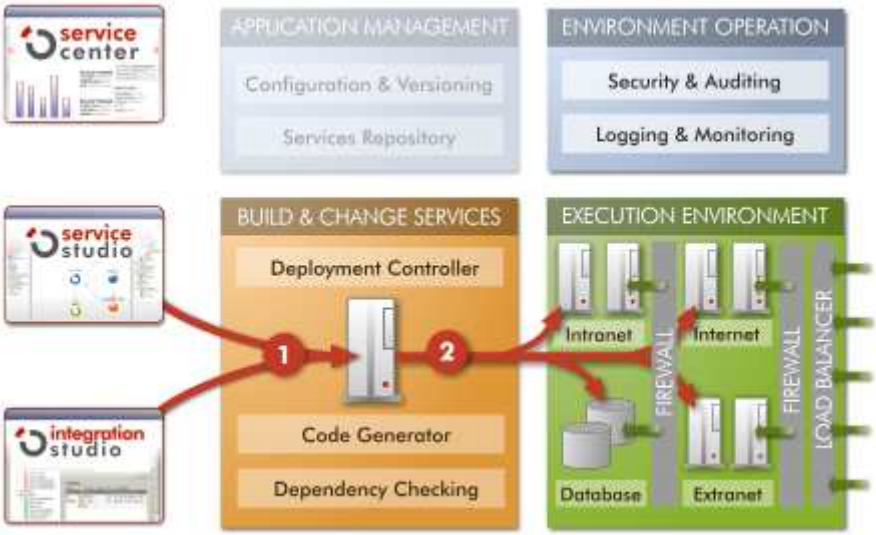


Figura 3 : Processo de publicação e execução **[23]**

1.1.4 Conclusão

Através da descrição anterior consegue-se perceber que uma aplicação em OutSystems não só é desenvolvida mais rapidamente que pelos processos tradicionais, permitindo atingir o *time-to-market* desejado, como é bastante flexível a qualquer alteração que surja.

O simples facto de todo o desenvolvimento nesta plataforma ser feita visualmente através de *drag'n'drop* torna a sua utilização muito simples, e permite a compreensão do que está a ser desenvolvido por pessoas com poucos conhecimentos tecnológicos.

Estes factores estão de acordo com as necessidades actuais das organizações que necessitam de adequar os seus sistemas o mais rapidamente possível às constantes alterações às condições do mercado.

Em comparação com as BPMS's apresentadas abaixo, esta plataforma é mais simples e consegue ser ainda mais flexível devido à nova metodologia de desenvolvimento que promove, no entanto esta não é um real concorrente às BPMS's pois têm objectivos diferentes.

2. Contextualização

A presente secção irá descrever o estado da arte da execução de processos de negócio e de algumas das ferramentas de processos de negócio actuais.

2.1 Gestão de Processos de Negócio

Segue-se uma breve descrição do estado da arte da Gestão de Processos de Negócio para melhor se perceber de onde surge a necessidade de execução de Processos de Negócio.

2.1.1 Gestão de Processos de Negócio

O interesse nos processos de negócio surge da necessidade de organizar operações de negócio, consolidar organizações e consequentemente reduzir custos, reflectindo o facto de os processos de negócio serem considerados a unidade básica de valor de uma organização.

Os processos de negócio existem desde sempre, embora tenham sido referidos por outros termos: procedimentos, actividades de trabalho, workflows, entre outros. Estes representam a forma como o trabalho é realizado e desta forma podem existir independentemente de qualquer tecnologia. Não é de estranhar que a Gestão de Processos de Negócio tenha tido origem nos anos 20 [33], bastante antes dos recentes avanços tornados possíveis pela era digital.

A evolução da Gestão de Processos de Negócio deu-se através de três frentes de onda:

1. A **primeira** teve início nos anos 20 e foi dominada pela *"Teoria da gestão" (Theory of Management)* de Frederick Taylor [33]. Os processos encontravam-se implícitos nas práticas de trabalho e não possuíam qualquer forma de automatismo.
2. Na **segunda**, correspondente à última década, os processos eram muitas vezes alterados de acordo com uma reestruturação de toda a organização. Eram utilizados ERPs (Enterprise Resource Planning systems), que eram soluções flexíveis até à sua instalação e rígidas após esta [33]. Estas reestruturações eram lentas e uma vez terminadas já se encontravam desactualizadas face às necessidades de negócio. Surgiu ainda uma tentativa de integrar ferramentas de workflow nos ERPs de maneira a torná-los mais flexíveis. Provavelmente um sistema de workflow por si só teria mais probabilidades de sucesso [5].
3. Na **terceira**, que corresponde ao estado actual, os processos de negócio são os blocos para a construção de qualquer sistema automático de negócio. A alteração é o objectivo a ser atingido pois para a Gestão de Processos de Negócio a possibilidade de alterar é mais valorizada que apenas a possibilidade de criar. Tendo isto em conta, é através da gestão de processos de negócio que as cadeias de valor são

monitorizadas e continuamente melhoradas. O feedback, a agilidade e a adaptabilidade são as palavras que compõem esta terceira onda. É nesta fase que surgem as *Business Process Management Suites (BPMS)*.

O estado actual do desenvolvimento informático permitiu que esta evolução tivesse lugar e culminasse nos dias de hoje.

Os *WorkFlow Management Systems* (WFMS) foram os primeiros sistemas com o objectivo de definir e gerir o fluxo de trabalho das organizações [34]. Contudo, estes ofereciam uma aproximação aos documentos e às intervenções humanas, descurando os sistemas, que são parte integrante das organizações e seus processos. Assim surgiram os BPMS que permitem endereçar actividades humanas ou orientadas a sistemas, e conseguem modelar todas as características inerentes aos processos.

Devido à forte componente de integração presente nestas ferramentas, BPM é muitas vezes confundido com *Enterprise Application Integration* (EAI). Uma vez que os sistemas de uma organização estão presentes nos seus processos de negócio, o BPM necessita de os integrar de forma a atingir a fácil gestão desses processos. Assim a integração é o meio para que a Gestão de Processos de Negócio possa atingir os seus objectivos, o que não invalida que as ferramentas EAI actualmente suportem processos de negócio.

O BPM oferece ao negócio a capacidade de adaptar os seus processos às condições actuais do mercado e a futuras adaptações, sem as colossais actividades de reengenharia a que assistíamos até hoje. Pode dizer-se que é o passo para a agilidade e adaptabilidade do *Business Process Reengineering* (BPR), pois as BPMS são vistas como as ferramentas que ajudam na tradução dos processos *"As Is"* para os processos "*To Be*" [32].

Deste modo as principais vantagens oferecidas pelo BPM são:

- Uma forma de passar *directamente* da visualização do que tem de ser feito (e entendido pela gestão) para um sistema capaz de o fazer;
- A agilidade para alterar processos em uso ou desenho sem o atrito normal causado pelas limitações das Tecnologias de Informação (TI);
- A capacidade de ligação, colaboração e integração entre processos;
- Uma plataforma para a partilha de processos de negócio entre sistemas, pessoas e parceiros;

2.1.2 Ciclo de Vida dos Processos de Negócio:

Para melhor entendermos o contexto em que surge a Execução de Processos de Negócio, convém entendermos o que é um processo e o seu ciclo de vida.

"A business process is the complete and dynamically coordinated set of collaborative and transactional activities that deliver value to customers." [33]

Desta forma um processo pode ser visto como um conjunto de actividades que produzem valor para a organização. Além disto:

- Facultam serviços à organização;
- Permitem que os objectivos organizacionais sejam atingidos;
- São vistos como a unidade base de valor da organização [38];

O ciclo de vida de um processo é em si um processo. Este encerra em si as actividades presentes na Figura 4.

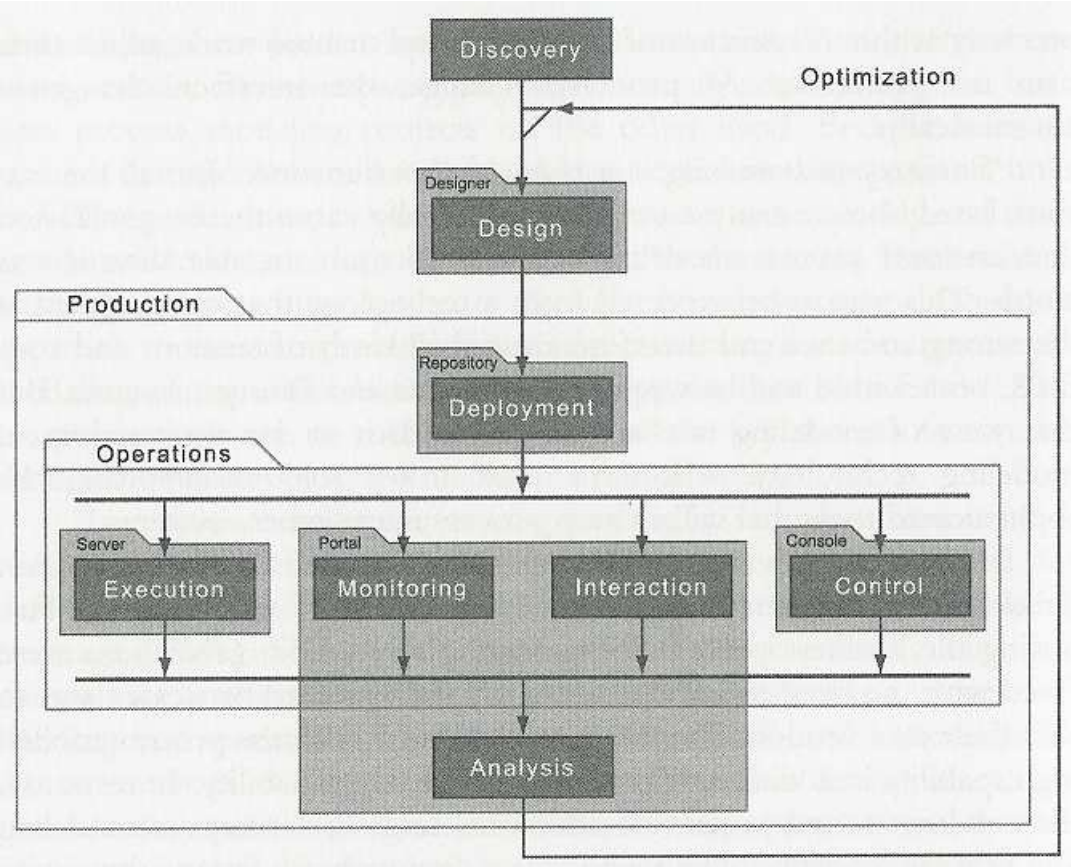


Figura 4: Processo de ciclo de vida dos processos **[33]**

Discovery: A fase de descoberta pretende clarificar como as actividades inerentes ao negócio são feitas. Os fluxos de eventos, informação e controlo são capturados tendo em conta a perspectiva dos vários participantes. É obtida uma visão clara de como os processos existentes funcionam interna e externamente, quais as suas interacções com outros processos, e onde recaem as responsabilidades que lhes estão subjacentes.

Design: A fase de desenho, também designada de modelação, tem como objectivo modelar os processos baseando-se na informação descoberta na fase anterior. As actividades, os participantes, as regras, as relações e interacções são os conceitos relevantes para o desenho do processo. Nesta fase o processo é submetido a várias transformações e reestruturações com o objectivo de o optimizar.

Deployment: É na fase de instalação que os processos desenhados na fase anterior são instalados. Os recursos necessários têm de ser reservados pois nesta fase as actividades do processo são delegadas nos vários participantes e sistemas. Com a terceira onda pretende-se que a fase de instalação seja rápida e fácil, com praticamente todas as actividades automatizadas.

Execution: A fase de execução tem como objectivo garantir que o processo é executado por todos os seus participantes (pessoas, sistemas, organizações e outros processos). Um sistema de gestão de processos gere o estado dos processos e suas interacções com os participantes. Este sistema de gestão de processos, esconde do utilizador de negócio os pormenores tecnológicos e é responsável pelo controlo e persistência da informação gerada durante a execução.

Interaction: A fase de interacção corresponde ao uso de portais que permitem aos utilizadores interagirem com o processo. Inclui-se a gestão da interacção existente entre o trabalho manual e a automatização, como por exemplo, toda a gestão e reserva de tarefas para os trabalhadores ou mesmo a gestão da introdução de dados necessários ao processo.

Monitoring: Com a fase de monitorização pretende-se manter uma visão sobre a performance geral do processo. Actualmente recorre-se a soluções *Business Activity Monitoring (BAM)* [15] que oferecem informação em tempo real sobre o processo, retirada dos vários sistemas.

Control: Esta fase encontra-se fortemente ligada à anterior pois as acções de controlo têm por base o que foi previamente monitorizado. Estas acções pretendem manter os processos a correr correctamente de um ponto de vista tecnológico e de utilização de recursos. É aqui que são tratados quaisquer erros e excepções que possam surgir e feitas pequenas alterações aos processos, como por exemplo, alteração de participantes.

Analisys: A fase de análise é responsável por medir a performance dos processos de forma a descobrir estratégias de optimização ou meios de inovação. São efectuadas interrogações à informação gerada pelo processo e ao processo em si. Com base nesta informação o processo é avaliado tendo em conta utilizações passadas e possíveis utilizações futuras. Os processos são sempre avaliados tendo em conta os objectivos de negócio. Esta fase recorre à simulação para analisar casos *"what-if "* e validar os processos e permite descobrir novas oportunidades bem como testar novos desenhos de processos.

2.1.3 Modelação e Execução

Como referido anteriormente, o BPM [17] propõe-se oferecer um elevado grau de flexibilidade nas organizações e no modo como estas actuam. Vamos agora ver mais detalhadamente como essa flexibilidade é atingida.

Após descoberto e modelado um processo tem de ser validado pelos responsáveis de negócio e consequentemente estes necessitam de compreender o que estão a avaliar. Neste contexto, surgiram várias linguagens de modelação que permitem a modelação visual de qualquer tipo de processo de negócio, possibilitando assim uma fácil transferência de informação entre os vários responsáveis.

As linguagens de modelação mais utilizadas e conhecidas são:

- ***Business Process Modeling Notation (BPMN)*** [39]: desenvolvido para a modelação de processos de negócio.
- ***UML Activity Diagrams*** [27]: desenvolvido para a modelação de fluxos de actividades, nos quais se enquadram os processos de negócio.
- ***Petri Nets*** [36]: utilizadas para modelação de workflows.

Estas linguagens, só por si, não produzem grandes avanços para além do aumento da facilidade de transferência de informação sobre os processos. Contudo cada vez mais linguagens de modelação suportam a tradução para código executável ou para linguagens de execução de processos de negócios.

Na Tabela 1 apresentam-se exemplos de traduções actualmente suportadas [14], [25]:

Tabela 1 : Tradução Modelação – Execução

Ling. Modelação	Ling. Execução
BPMN	BPEL ou BPML
UML-AD	BPEL
Petri Nets	PNML

Deste modo, a modelação de processos já produz grande parte do necessário para a sua execução, diminuindo o tempo entre o desenho e instalação. Eliminam-se assim os enormes tempos de desenvolvimento de sistemas necessários para colmatar necessidades urgentes.

A modelação visual, aliada à execução, permite atingir a flexibilidade e adaptabilidade necessárias às organizações.

Para suportar o BPM nas organizações surgiram novas ferramentas chamadas *Business Process Management Suites* (BPMS) que englobam várias aplicações que suportam todas as fases

inerentes aos processos de negócio, da modelação à execução, com capacidades para monitorização e apoios para a optimização.

Estas ferramentas permitem toda a gestão de processos de negócio numa organização. No entanto, existem umas melhores que outras e actualmente nenhuma delas atinge todos os objectivos propostos pelo BPM.

2.2 Execução de Processos de Negócio

A secção seguinte pretende mostrar algumas das tecnologias actuais na área dos Processos de Negócio e *Workflow*, no entanto não pretende ser de forma alguma uma descrição exaustiva destas mesmas tecnologias.

2.2.1 BPEL

O BPEL *(Business Process Execution Language)* [4], [16], [37] surgiu como linguagem para a definição e execução de processos de negócio.

Esta linguagem, baseada em XML e Web Services, foi inicialmente desenvolvida pela *Microsoft*, *IBM* e *BEA Systems*, surgiu do trabalho desenvolvido pela *Microsoft* no XLANG, inspirado em programação estruturada e do trabalho da *IBM* no WSFL *(Web Services Flow Language)* que abordava um processo como um grafo directo. O BPEL tenta integrar ambas as aproximações.

Actualmente o BPEL encontra-se ao encargo do comité técnico OASIS [19], que publicou originalmente esta linguagem em Agosto de 2002 e lançou a versão 2.0 [20] em Dezembro de 2005 com várias revisões até ao momento.

O BPEL suporta a definição de dois tipos de processos: processos abstractos e processos executáveis. Os processos abstractos são processos não executáveis, úteis para a especificação de protocolos de comunicação entre elementos diferentes sem revelar comportamentos internos. Os processos executáveis são bastante mais detalhados, especificam a ordem de execução das actividades que os compõem, bem como os participantes, mensagens trocadas entre estes e finalmente um conjunto de tratamento de erros e excepções.

A Tabela 3 no anexo 1.1 apresenta uma tabela com uma listagem não exaustiva dos elementos da linguagem BPEL da versão 1.1 (Versão suportada pela maioria das ferramentas) para que se possa ter uma noção do que esta linguagem permite.

A Figura 48 no anexo 1.2, apresenta um exemplo de um processo definido em BPEL.

As ferramentas que suportam BPEL podem seguir duas aproximações diferentes:

- **Importação:** Os processos são importados nesta linguagem que apenas é usada como meio de definição de processos, já que antes da execução esta é traduzida para a linguagem interna da ferramenta. Por exemplo o *BEA AquaLogic BPM Suite* (Apresentado mais à frente) funciona desta forma.
- **Interpretação:** As ferramentas possuem um *BPEL Process Engine* que executa os processos através da interpretação desta linguagem. Exemplo de um *BPEL Process Engine* em [2].

Outras linguagens deste género surgiram mesmo antes do BPEL, como por exemplo o BPML suportado pelo BPMI. Contudo o facto do BPEL ter o apoio da indústria (*Microsoft, IBM, BEA Systems*) fez com que este se tornasse a norma mais usada para execução de processos de negócio, sendo suportada pela maioria das ferramentas actualmente no mercado e superando toda a concorrência.

Apesar de ser a mais usada actualmente, esta linguagem possui várias falhas:

- Não tem componente gráfica, o que dificulta a definição e percepção de processos nesta linguagem, o que leva a que cada ferramenta desenvolva a sua própria linguagem gráfica.
- É muito baixo nível para definir processos, aproximando-se mais a uma linguagem de programação do que uma linguagem para definição de processos.
- Não oferece suporte para a análise de processos, não conta com a necessidade de monitorizar os processos e não oferece qualquer meio para que tal seja suportado.
- Não tem suporte para integração pois não possui elementos para a transformação de protocolos entre sistemas.
- Não possui abstracções para actividades humanas e para os elementos associados a estas actividades como: papéis, objectos de trabalho e inboxes.

Como é natural, a indústria tenta solucionar estas falhas nas suas ferramentas o que levou a *IBM* e a *SAP* em Agosto de 2005 a proporem uma extensão ao BPEL denominada de BPEL4People [13] que incorpora tarefas humanas.

2.2.2 Petri Nets

As ferramentas de workflow muitas vezes possuem linguagens de modelação proprietárias. No entanto, quando se trata da execução, estas ferramentas pretendem ser reutilizáveis e assim necessitam de uma linguagem de definição bem conhecida e não ambígua que permita a análise das definições produzidas. Assim, adoptou-se o uso de *Petri Nets* pelas razões que se seguem:

- Apresentam uma linguagem simples, clara e intuitiva. São suportadas por uma definição visual. Os seus elementos básicos permitem a definição de qualquer tipo de processo.
- Têm uma forte base matemática que pode ser usada na sua implementação.

- São independentes de qualquer vendedor, não se baseando em qualquer produto ou tecnologia.

As *Petri Nets* surgiram nos anos sessenta do trabalho de Carl Adam Petri [36]. Segundo a definição clássica, as *Petri Nets* possuem dois tipos de nós: *places* (lugares) e *transitions* (transições). Estes nós são ligados através de *arcs* (arcos). A ligação de dois nós do mesmo tipo não é permitida.

A Figura 5 apresenta a definição formal das *Petri Nets* e a sua representação gráfica.

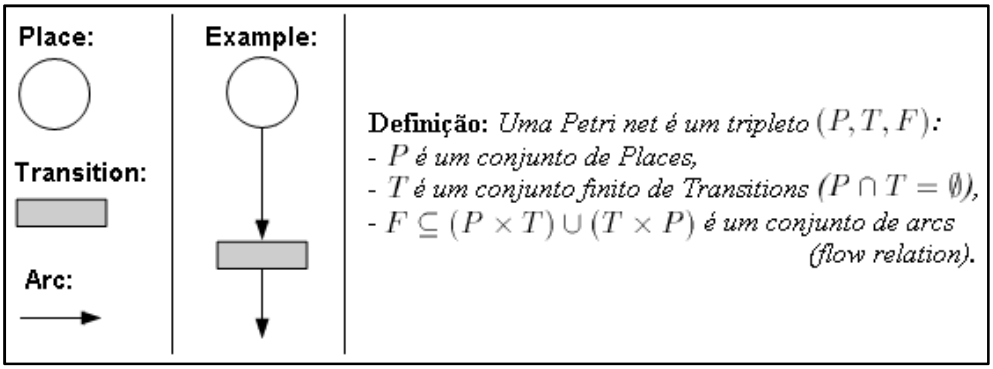


Figura 5 : Definição de Petri Nets **[36]**

Do ponto de vista da execução tem-se que os *tokens* (elemento que representa um fluxo de execução) permanecem nos *places* até que as *transitions* sejam disparadas. Quando uma *transition* é disparada esta consome um *token* de cada *place* que lhe está ligado e coloca um novo *token* em cada *place* a que esta se liga.

Qualquer implementação baseada nesta tecnologia necessita de uma forma de ligar os elementos anteriores às actividades inerentes aos processos de negócio. Por exemplo, Diogo Ferreira [6], [7] propõe que cada *place* tenha associado *actions* (acções) responsáveis pela execução das actividades de negócio e pela produção de eventos que irão disparar as *transitions* fazendo o processo avançar. Estas *actions* fornecem bastante flexibilidade a estes sistemas, pois podem ser implementadas como *Web Services*, Interacção Humana, Integração com outros sistemas, etc..

Propõe também que o *workflow engine* e as *actions* ofereçam um conjunto de interfaces de forma a permitir a gestão dos processos, mais concretamente para permitir às *actions* enviarem eventos para o *workflow engine (INotifiySink)* e permitir a este chamar as *actions* associadas aos *places (IAction)*.

A Figura 6 exemplifica esta proposta de implementação de *Petri Nets*.

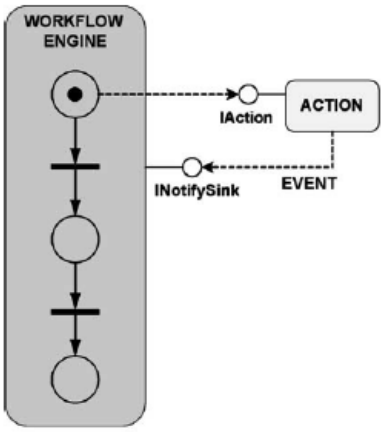


Figura 6 : Petri Nets Implementation **[7]**

De maneira a ser possível estender as capacidades do *engine* é necessário que este possa comunicar com as outras componentes as alterações que surgiram ao processo, quer alterações à sua estrutura, quer alterações ao seu estado de execução (eventos entre outros). Assim é proposto que todos os componentes ofereçam também uma interface ao *workflow engine (INotifySink)* que este utilize para notificar os outros componentes sobre alterações que surjam. Esta interface tem de ser suficientemente rica para permitir a notificação de qualquer tipo de alteração que possa surgir. A Figura 7 exemplifica.

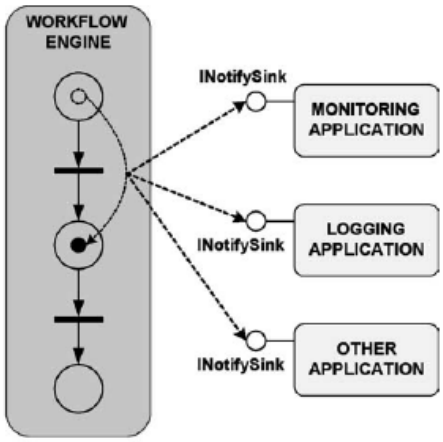


Figura 7 : Petri Nets Implementation **[7]**

Desta forma podemos ter implementado sobre a tecnologia *Petri Nets* um *workflow engine* capaz de execução de processos de negócio e capaz de integração. A possibilidade de adicionar componentes oferece-nos meios para monitorizar e recolher dados para posterior análise e optimização dos processos, tal como a maioria das ferramentas anteriores baseadas em BPEL.

2.2.3 WAPI

A *WAPI (Workflow APIs Interchange format)* [11] é uma norma de interfaces desenvolvida pelo *WFMC (WorkFlow Managemente Coalition)* para promover a reutilização e interoperabilidade entre sistemas de *Workflow*.

Como se pode ver pela Figura 8, esta norma permite separar os sistemas de gestão de *workflow* em vários componentes recorrendo a cinco interfaces que permitem a interoperabilidade entre estes. Bem no centro desta arquitectura encontra-se o *Workflow Enactment Service* que fornece o ambiente de execução no qual os processos são instanciados. Este componente é também responsável por interagir com os recursos externos necessários à execução do processo.

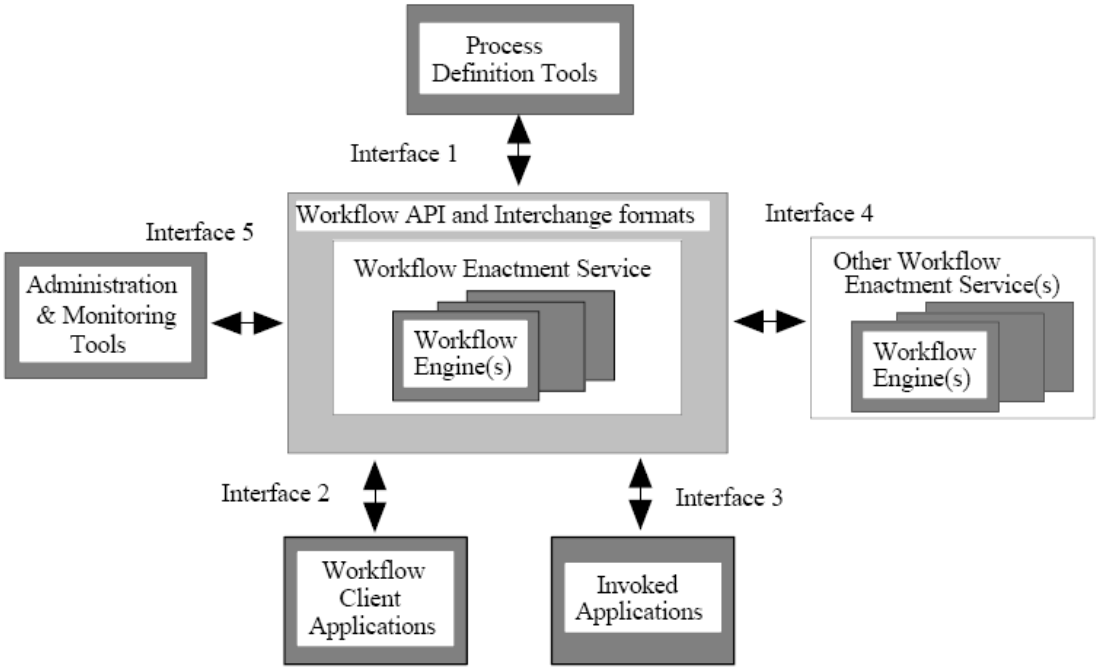


Figura 8 : Separação dos sistemas de gestão de *workflow* segundo a norma WAPI **[11]**

Segue-se uma breve explicação das interfaces e dos componentes com que estas interagem:

- **Interface 1:** Esta é a interface entre a componente de modelação e o *Workflow Enactment Service*. Através desta interface são trocadas as definições dos processos modelados que irão ser executados.
- **Interface 2:** Esta é a interface entre a componente responsável por gerir a interacção humana *(Workflow Client Applications)* e o *Workflow Enactment Service*. Através desta interface são trocados dados como as actividades a serem realizadas pelos utilizadores, e os dados introduzidos no decorrer dessas actividades, necessários à execução do processo.
- **Interface 3:** Esta é a interface entre as aplicações a serem usadas pelo *Workflow Enactment Service* e ele próprio. É através desta interface que são chamadas todas as

aplicações externas ao sistema mas necessárias para a execução do processo, *Web Services* por exemplo. Para além de ter de permitir as chamadas destas aplicações tem também de ter em conta o seu retorno.

- **Interface 4:** Esta é a interface entre o *Workflow Enactment Service* e outros *Workflow Enactment Service*s que possam ser usados para ajudar na execução dos processos. Através desta interface passa informação sobre a distribuição da execução pelos vários *Workflow Enactment Services* que podem seguir várias aproximações [11].
- **Interface 5:** Esta é a interface entre as componentes de monitorização e administração e o *Workflow Enactment Service*. Nesta interface devem passar os dados de execução dos processos de forma a serem apresentados na componente de monitorização. Deve ainda permitir a gestão de processos, como por exemplo, o cancelamento ou suspensão da execução.

Esta norma promove a utilização de componentes de várias origens num único sistema, oferecendo assim a possibilidade de criar sistemas mais adequados à necessidade de cada organização sem a obrigatoriedade de estar ligado a um único vendedor.

No entanto, a principal vantagem de um sistema que implemente esta norma não é apenas a liberdade de utilização de componentes, mas sim a facilidade que oferece no que respeita à interoperabilidade entre sistemas. Permitindo uma fácil comunicação entre os componentes dos sistemas de gestão de processos, atinge-se também uma melhor gestão destes.

2.2.4 Outras Tecnologias

Existem ainda outras tecnologias que por serem semelhantes às anteriores ou por serem pouco usadas na indústria não serão abordadas com muito pormenor.

- **A geração de código** [2] é um dos meios usados para executar processos de negócio. Tendo como base um processo (numa linguagem de definição de processos pré definida), pode-se gerar código que permita a execução dos processos previamente descritos, especialmente se o código for gerado para uma linguagem orientada aos objectos. Este é um dos meios que os *process engines* actuais utilizam para interpretar as linguagens de processos.
- **O XPDL** *(XML Process Definition Language)* [10], [35] é uma linguagem de definição de processos de negócio concorrente do BPEL e BPML desenvolvida pelo WFMC *(WorkFlow Management Coalition)* e tem como principal objectivo a troca de processos entre ferramentas de modelação. Como apresentado na Figura 8 é a linguagem usada para trocar definições de processos na interface 1 da arquitectura WAPI.
- **O BPML** *(Business Process Modeling Languange)* [10] é uma linguagem de definição de processos de negócio baseada em XML, concorrente do BPEL. Esta linguagem foi desenvolvida pelo BPMI e pretende descrever a representação estrutural de um processo e

a sua semântica de execução. Tal como acontece com o BPEL, o objectivo do BPML é a execução de um processo em XML, elemento a elemento, num *process engine*. Embora tenha surgido primeiro que o BPEL não teve o apoio da indústria tal como o seu concorrente.

- **O Pi-Calculus** [10] é uma linguagem formal para definir a comunicação de processos concorrentes, processos estes que podem ou não ser processos de negócio. Esta linguagem tem raízes académicas e fortes bases algébricas, o que torna difícil a sua adopção por parte dos analistas de negócio. Apesar da sua complexidade, alguma literatura sobre BPM afirma que esta linguagem está na base de outras linguagens como o BPML e o BPEL.
- **O PNML** *(Petri Net Markup Language)* [12] é uma linguagem baseada em XML que permite definir *Petri Nets*. Esta linguagem é usada por algumas ferramentas de *process mining* como meio de *input* de definições de processos.

2.3 Ferramentas Actuais de Execução de Processos de Negócio

Nesta secção serão analisadas três ferramentas BPM que se espera exemplifiquem a actualidade da oferta nesta área.

Os pontos apresentados serão a estrutura das ferramentas (para compreensão da adaptabilidade ao ciclo de vida dos processos), e a aproximação que cada ferramenta tem à execução de processos de negócio.

2.3.1 IBM WebSphere BPM Suite v6.0

A *IBM WebSphere BPM Suite* é uma das ferramentas mais conhecidas nesta área e pretende incluir suporte não só para a gestão de processos, mas também para a integração de sistemas. Possui uma arquitectura orientada aos serviços (SOA) que é apresentada na Figura 9.

Esta ferramenta integra quatro componentes:

- ***Business Modeler:*** Aborda a parte da modelação. Permite modelar fluxos de processos, recursos, custos, dados e indicadores de performance para serem usados nas fases de análise e optimização.
- ***Integration Developer:*** Responsável pela integração. Suporta a importação de processos em BPEL e outros objectos de negócio a partir do *Business Modeler* carregando-os no *Process Server* para que estes sejam executados.
- ***Process Server:*** É a base da arquitectura SOA da plataforma. Possui um motor de orquestrações BPEL, um motor de *business rules* e um ambiente de execução de outros componentes.

- ***Business Monitor:*** Responsável pela monitorização e optimização. Utiliza dados recolhidos durante a execução e oferece ferramentas que baseando-se nas métricas definidas no *Business Modeler*, permitem uma análise histórica e em tempo real (recorrendo a *dashboards*) da performance.

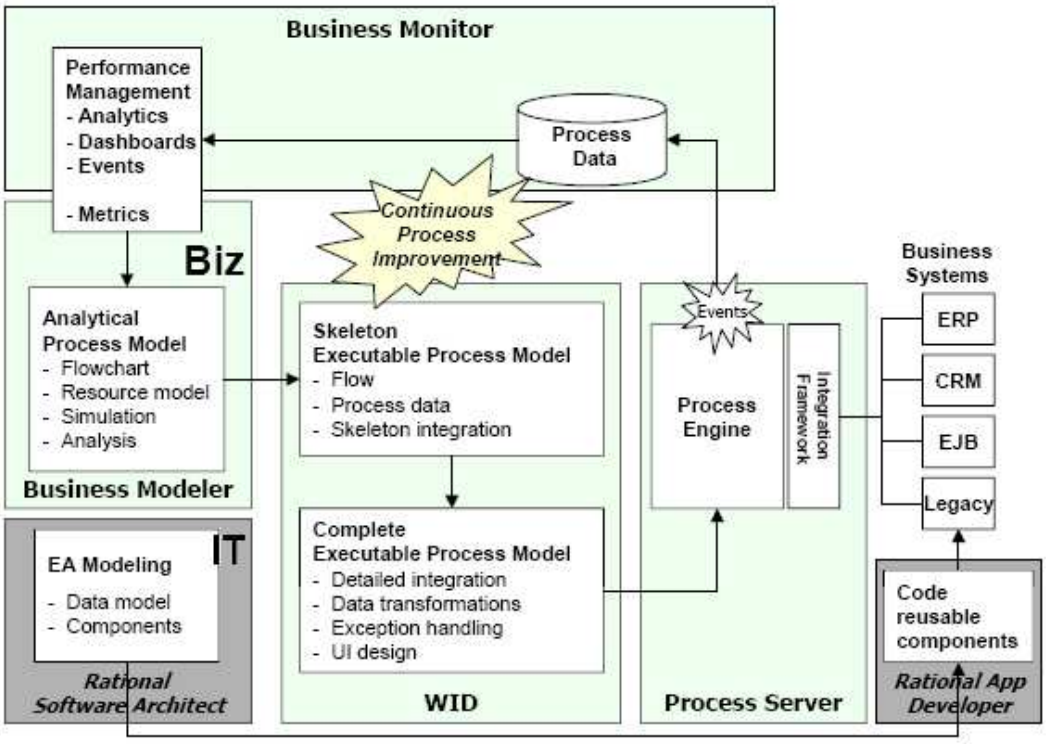


Figura 9 : *IBM WebSphere Suite Architecture* **[30]**

A componente de execução *(Process Server)*, apesar do seu nome foi construída de modo a integrar *service components* e não a automatizar processos de negócio. Estes são vistos como um tipo de *service component*.

São suportados dois tipos de processos: os processos de negócio comuns e as máquinas de estado de negócio. Os processos de negócio representam a aproximação comum em cada passo corresponde a uma actividade BPEL. As máquinas de estado de negócio são mais adequadas à gestão de documentos em que a acção a tomar depende do estado actual. Estas possuem estados, e o avanço entre estes mesmos estados é baseado em eventos. No entanto, após a definição, estas máquinas de estado são também traduzidas para BPEL de maneira a serem executadas.

Para contornar o problema que o BPEL levanta (falta de suporte a actividades humanas), é oferecido um componente chamado *Human Tasks* que permite definição de actividades humanas como *Web Services*, pois estas exportam uma interface que permite invocá-las, permitindo assim a sua utilização em BPEL.

Esta ferramenta é das poucas a suportar BPEL 2.0 (norma da indústria para a execução de processos de negócio) em toda a sua estrutura interna.

As suas componentes abrangem todo o ciclo de vida dos processos, do desenho à optimização, oferecendo diferentes vistas consoante o utilizador.

Com esta ferramenta consegue-se uma boa gestão dos processos de negócio, contudo esta segue uma abordagem mais próxima da integração de sistemas do que da gestão de processos. Assim, consegue implementar qualquer processo de negócio pois as suas ferramentas de integração oferecem elementos para integrar com qualquer tipo de sistema ou qualquer tipo de processo, mas torna mais lenta, complexa e difícil a gestão do processo implementado.

2.3.2 BEA AquaLogic BPM Suite v5.5

A *AquaLogic BPM Suite* da *BEA Systems*, é uma ferramenta com suporte para gestão de processos de negócio e integração de sistemas. A Figura 10 mostra a arquitectura desta ferramenta.

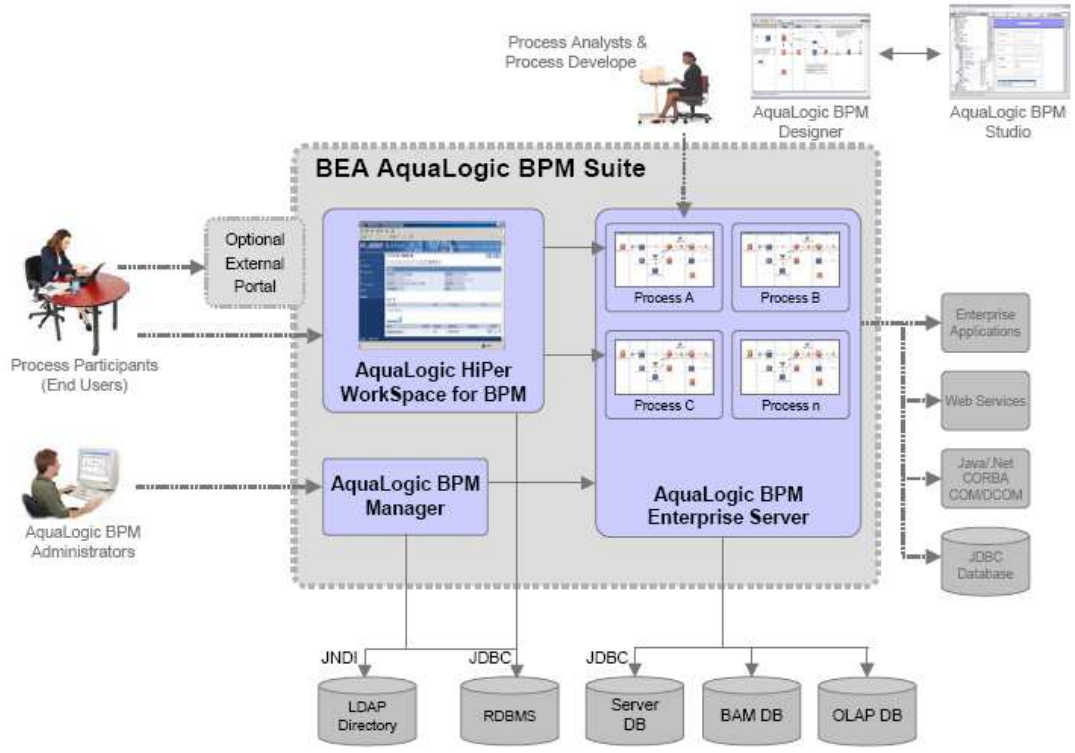


Figura 10 : *BEA AquaLogic BPM Suite Architecture* **[29]**

Esta ferramenta integra as seguintes componentes:

- ***BPM Designer:*** Componente visual para a modelação de processos (pelo analista de negócio) com suporte para BPMN e UML.

- ***BPM Studio:*** Componente para integração, onde se desenvolvem os adaptadores para integrar com outros sistemas. É também neste componente que se desenvolvem as partes baixo nível dos processos.
- ***BPM Enterprise Server:*** Componente responsável por orquestrar todos os processos e seus recursos. Executa processos definidos nas componentes anteriores. Permite a importação de processos em BPEL. Inclui o *Hiper Workspace* for BPM um subcomponente responsável pelo processamento de actividades com interacção humana.
- ***BPM Manager:*** Este componente oferece uma consola que permite o acesso a informação em tempo real sobre os processos e a manipulação destes.
- ***BPM DashBoard:*** Oferece aos utilizadores de negócio e administradores todos os dados e historial dos processos que se encontram em execução no *BPM Enterprise Server*. Dá assim suporte à análise da performance dos processos.

Esta ferramenta é orientada aos serviços, embora tecnologicamente assente em XPDL.

Um processo em *AquaLogic* é uma orquestração de actividades, que representam serviços de negócio: interacção humana, lógica de negócio automática, integração com sistemas e criação de sub-processos.

Cada actividade contém uma ou mais tarefas. As implementações das tarefas são denominadas de métodos e não são mais que *scripts* desenvolvidos numa linguagem de *scripting* proprietária da *AquaLogic* chamada a *Fuego Business Language*. Estes métodos são normalmente invocações a objectos de negócio que podem ser: lógica de negócio, interacção humana ou integração com sistemas. Estes encontram-se no catálogo de componentes e são especificados no *Component Manager* interno ao *BPM Studio*.

O *Enterprise Server*, responsável pela execução, orquestra os processos tendo em conta as suas definições e os *scripts* das suas actividades. É também responsável pela gestão dos *Business Objects*, pela integração com outros sistemas, e pela recolha de informação para uma base de dados para que esta possa ser usada pelas componentes de análise e optimização.

Esta ferramenta suporta a importação de várias definições usadas por outras ferramentas do mercado e BPEL, embora não execute os processos sobre esta linguagem.

Abrange todo o ciclo de vida dos processos, oferecendo componentes para todas estas fases adequadas aos seus respectivos utilizadores.

Esta ferramenta tem uma aproximação mais virada para a gestão de processos de negócio que a anterior. Tem uma forte componente de integração suportada por adaptadores que permitem integrar com qualquer tipo de sistema ou tecnologia, possibilitando o uso de introspecção para o conseguir.

2.3.3 Savvion BusinessManager v6.5

O *Savvion Business Manager* embora suporte integração de sistemas, coloca-se como uma ferramenta completa de gestão de processos de negócio. A Figura 11 mostra a arquitectura desta ferramenta.

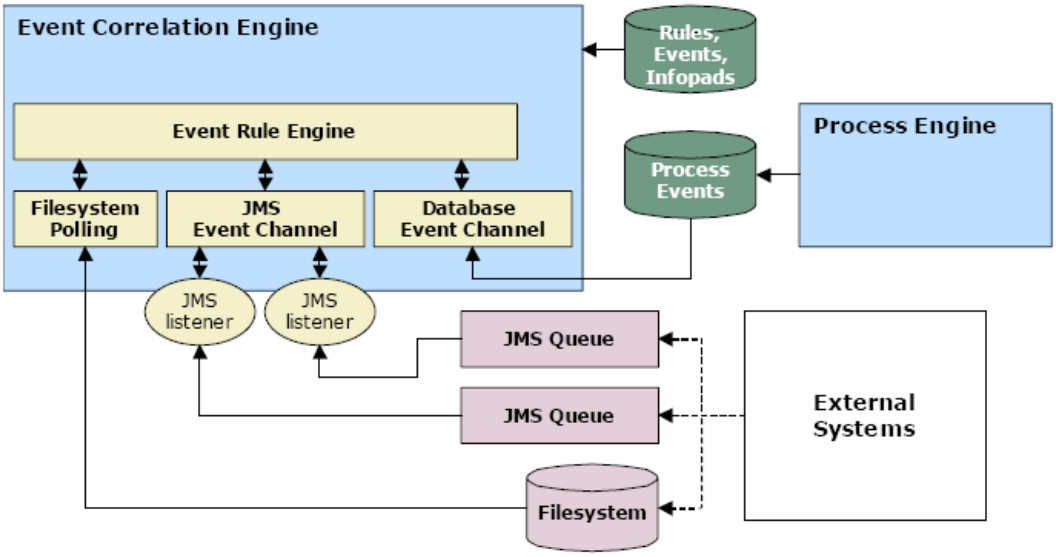


Figura 11 : *Savvion BusinessManager Suite Architecture* **[31]**

Esta ferramenta integra as seguintes componentes:

- ***Process Modeler:*** Componente de modelação e simulação direccionada aos analistas de negócio. Usa BPMN como linguagem de modelação.
- ***BPM Studio:*** Componente de desenvolvimento dos processos tendo em vista a execução. Integra toda a funcionalidade do *Process Modeler* para fornecer um completo ambiente de desenvolvimento numa só aplicação.
- ***BPM Server:*** Executa os processos tratando de todo o tipo de actividades como por exemplo: actividades automáticas, interacção humana, gestão de eventos. Possui ainda capacidades de integração.
- ***BPM Portal:*** Componente através da qual se interage com os processos. A interacção humana com os processos, bem como a sua administração é efectuada neste componente. Oferece uma colecção de aplicações optimização e monitorização das quais fazem parte *scorecards*, *dashboards* e alertas que permitem a monitorização da execução dos processos.

Um processo, é um conjunto de *worksteps* (actividades do *Savvion*) associados a um responsável. Um responsável pode ser um utilizador, um adaptador para outro sistema, um *Web Service* ou mesmo um sub-processo. Estes *worksteps* permitem a definição de acções à entrada e/ou saída destes definidos em *JavaScript*.

Durante a execução de um processo, sempre que se chega a um *workstep* é gerado um *workitem* para o responsável do respectivo *workstep* e só mediante a concretização desse *workitem* é que é permitido ao processo avançar.

A orquestração dos *worksteps* é efectuada sobre um *process engine* baseado na tecnologia J2EE, bastante poderoso, que permite a alteração da definição de instâncias de processos (todas as instâncias possuem uma cópia da definição do processo). É assim possível, para um utilizador, acrescentar em tempo de execução um passo a uma das instâncias.

As componentes desta ferramenta permitem uma gestão dos processos durante todo o seu ciclo de vida. As componentes desta plataforma separam o detalhe de implementação dos analistas de negócio, deixando essa dificuldade para os responsáveis de desenvolvimento.

Esta plataforma foi desenvolvida para a gestão de processos de negócio, dando grande importância aos seus componentes de apresentação de informação sobre a execução dos processos. É a única de entre as apresentadas que permite a flexibilidade de alterar instâncias dos processos em tempo de execução.

Tem uma forte componente de integração, necessária para que os seus processos possam interagir com outros processo, mas apesar disto o objecto central desta plataforma é o processo.

2.3.4 Comparação

A Tabela 2 apresenta uma comparação das ferramentas anteriores nos pontos mais relevantes segundo o BPM.

Embora todas as ferramentas mencionadas abordem completamente o ciclo de vida dos processos, o *Savvion Business Manager* é o mais virado para a Gestão de Processos de Negócio. As outras ferramentas também permitem essa tarefa, contudo são ferramentas desenvolvidas no âmbito da integração de sistemas e posteriormente adaptadas às necessidades da Gestão de Processos de Negócio.

O *IBM WebSphere* é a única ferramenta das apresentadas cuja arquitectura de execução se baseia numa norma. Entre as outras ferramentas, o *BEA AquaLogic* também suporta esta norma, embora não baseie a sua arquitectura nela.

Finalmente pode-se concluir que embora sigam aproximações diferentes, a execução em todas as ferramentas apresentadas se baseia na gestão de orquestrações de actividades.

A principal conclusão que se pode retirar é que as ferramentas actuais, ainda não oferecem a flexibilidade e agilidade promovida pelo BPM. Estas ferramentas fornecem grandes melhorias neste campo quando comparadas com os sistemas anteriores, no entanto estão ainda longe do que seria pretendido, continuando a perder-se muito tempo nos processos de desenvolvimento.

Algumas das ferramentas oferecem uma visão mais virada para a integração de sistemas do que para a gestão de processos. É verdade que actualmente a maior dificuldade na implementação deste tipo de sistemas se prende com a integração com outros sistemas, ainda assim, uma vez instalado o sistema, são as componentes de gestão da ferramenta que devem sobressair, pelo que estas não devem ser minimizadas.

Tabela 2 : Comparação das ferramentas de BPM

Ferramentas	Modelação	Execução	Ciclo de Vida	Normas	Aproximação
IBM WebSphere	Linguagem Proprietária	*Process engine* que executa orquestrações em BPEL.	Aborda todo o ciclo de vida.	Executa processos em BPEL	Mais virada para a integração de sistemas que para a Gestão de Processos.
BEA AquaLogic	Linguagem Proprietária Suporta BPMN e UML	Motor que gere orquestrações de actividades.	Aborda todo o ciclo de vida.	Definições em XPDL. Permite a importação de processos em BPEL	Mista entre a Gestão de Processos e a Integração de sistemas. As suas componentes ainda se baseiam muito na integração.
Savvion Business Manager	BPMN	Motor em J2EE, gere orquestrações de actividades. É a única que permite alterações ao processo em execução.	Aborda todo o ciclo de vida.	A modelação é feita em BPMN.	Mais virada para a Gestão de Processos de Negócio, embora a sua componente de integração seja bastante completa.

3. Problema

O aumento da tecnologia foi provavelmente o principal motivador das diferenças actuais no negócio, a facilidade de comunicação e movimentação actuais levaram a um alargamento das fronteiras negociais tornando os mercados tradicionais em mercados globais, onde a concorrência é muito mais variada e feroz.

3.1 Ambiente de Negócio

Nos dias que correm, o ambiente de negócio é influenciado por uma grande pressão existente sobre as organizações. Estas apontam os seus esforços para conseguirem acompanhar as incessantes mudanças às condições do mercado. A capacidade ou incapacidade de adaptação a estas condições pode ditar a sobrevivência de uma organização, pois um desalinhamento com as necessidades do mercado cria na organização uma incapacidade para a criação de valor.

Estas constantes alterações do mercado, da concorrência e dos clientes, levou a que as organizações procurassem uma flexibilidade máxima para criar respostas atempadamente e com valor, que irão proporcionar o sucesso, ainda que temporário, da respectiva empresa. Uma posição estável obriga uma organização a uma continuada reacção às condições que vão surgindo ou alterando-se.

Os factores anteriores levaram a que as organizações reestruturassem todas as suas actividades internas, passando a ter como unidade central não o produto ou serviço que desenvolvem, mas o “Processo de Negócio” que o permite atingir. Desta forma, a adaptação das organizações às alterações das condições de mercado passa pela adaptação dos seus processos de negócio.

O interesse na Gestão de Processos de Negócio aumentou bastante, pois passou a ser uma actividade crucial na gestão das organizações. No entanto, sem a capacidade de modelar, de executar e monitorizar os processos, é mínima a vantagem que estes trazem a qualquer organização. As alterações necessárias aos processos são criadas através da modelação que uma vez completa produz processos que idealmente estariam prontos para serem colocados em execução, ficando desta forma a organização adaptada às necessidades.

Para que a agilidade permitida pelos processos de negócio, descrita anteriormente, seja possível é necessário que toda a organização e seus sistemas estejam interligados permitindo que a alteração dos processos se reflicta por todos os sistemas que controlam a organização. Com todos os sistemas interligados, consegue-se adaptar os processos da organização pela modelação e adaptação da lógica inerente a estes processos e sistemas. Percebe-se assim a importância de

executar processos de negócio. Estes permitem agilizar as actividades e sistemas de uma organização, e a fácil alteração e posterior execução torna a empresa flexível a mudanças vindouras.

Neste sentido surgiu um novo conjunto de ferramentas, bastante usadas actualmente, para ajudar à gestão dos processos de negócio. Estas ferramentas as *Business Process Management Suites*, descritas em maior detalhe no capítulo anterior, oferecem apoio para todas as actividades ligadas à gestão de processos de negócio. As BPMS's oferecem componentes para a modelação dos processos, para a simulação de processos permitindo testar a eficiência destes, para a integração com os sistemas das organizações, para dar apoio à execução e monitorização dos processos.

No entanto, algumas destas ferramentas tendem a ser demasiado baixo nível, sendo sempre necessários elementos das TI para efectivar as alterações necessárias aos processos. Deste modo, a modelação e reestruturação dos processos de negócio são normalmente realizadas pelas pessoas do negócio, pois são estes que conhecem as necessidades e as respostas correctas que têm de ser levadas a cabo, em seguida elementos das TI ficam responsáveis por implementar a integração com os sistemas que darão suporte aos novos processos e ainda por programarem toda a lógica de negócio associada a estes. É obvio que não se pretende que sejam os elementos do negócio a desenvolver os sistemas de raiz, pois não é essa a sua função nas organizações, mas é importante que consigam no mínimo compreender os sistemas desenvolvidos para que possam efectuar uma real validação do que foi desenvolvido, para compreenderem se o que irão aprovar é ou não o desejado.

O estado actual destas ferramentas proporciona uma flexibilidade enorme quando comparada com a flexibilidade existente nas organizações alguns anos atrás, mas ainda assim apresentam algumas falhas, especialmente no que diz respeito ao modo de desenvolvimento. Idealmente deveria ser necessário pouco ou mesmo nenhum trabalho após a modelação de um processo até este estar disponível para execução, mas em muitos casos não é o que se verifica.

Temos então, que o principal problema encontrado prende-se com a necessidade que as organizações têm em usar e gerir os seus processos de negócio de forma ágil e flexível, necessidade que só é conseguida dando suporte para a execução dos processos de negócio. Outro problema, mais particular, tem em conta o facto das necessidades do negócio nem sempre estarem alinhadas com o que os elementos das TI, responsáveis pela reestruturação baixo nível dos processos, acabam por desenvolver.

3.2 OutSystems

Com os últimos avanços nas tecnologias de informação tem surgido um novo conjunto de ferramentas que permitem desenvolver aplicações através de metodologias ágeis. Estas ferramentas que não se apresentam como concorrentes das BPMS's resolvem ambos os problemas anteriores, pelo método de desenvolvimento proposto, mas levantam um novo. O facto de actualmente as organizações se encontrarem orientadas aos processos de negócio faz com que apesar de bastante

ágeis, estas ferramentas não estejam alinhadas com as organizações, pois estas são ferramentas de desenvolvimento e não apresentam suporte para processos de negócio.

A OutSystems é uma empresa que desenvolveu uma destas ferramentas. A plataforma OutSystems permite endereçar o que é esperado de um sistema empresarial hoje em dia, assim as aplicações desenvolvidas têm em conta a necessidade de mudança actual, são aplicações *built-to-change,* tal como é esperado que sejam os processos de negócio.

Perante isto pode-se afirmar que a plataforma OutSystems já oferece a flexibilidade e agilidade necessárias às organizações. As suas aplicações são desenvolvidas através de uma metodologia ágil que proporciona menos custos em relação à tradicional e produz os mesmos ou melhores resultados.

O facto de todo o desenvolvimento ser feito visualmente e as aplicações serem *web-based* torna-as mais fáceis de serem compreendidas por profissionais com poucos conhecimentos tecnológicos, ajudando a limitar outro dos problemas levantados pelas BPMS's.

Como já foi dito as organizações hoje em dia funcionam em torno de processos de negócio e necessitam de ferramentas que lhes permitam gerir os seus processos, ou seja, necessitam de ferramentas que estejam alinhadas com a sua estrutura interna. É neste ponto que a plataforma OutSystems apresenta a sua falha.

A falta de processos de negócio na plataforma OutSystems torna difícil para as organizações espelhar e acompanhar os seus processos nas aplicações desenvolvidas nesta plataforma. Embora forneçam a flexibilidade desejada, só com suporte para processos de negócio as organizações poderão encontrar nesta plataforma um suporte completo para os seus sistemas.

O facto anterior levou a que clientes da OutSystems começassem a questionar se de futuro a plataforma suportaria processos de negócio. Pode-se mesmo afirmar que as condições de mercado da OutSystems se alteraram e esta teve de encontrar resposta para essa alteração.

Ao incluir processos de negócio na sua plataforma, a OutSystems consegue que as aplicações desenvolvidas passem a estar de acordo com os processos das organizações. Consegue ainda que os processos beneficiem das vantagens já existentes na plataforma, como a agilidade acrescida e ambiente visual, todo o desenvolvimento dos processos de negócio é agora mais compreensível para os elementos do negócio, e passará a ser possível desenvolver processos de negócio facilmente e instalá-los para execução com um simples *click*, tal como já acontecia com as aplicações.

3.3 Motor de Execução Externo

Tendo em conta os factos anteriores, a forma mais rápida de os solucionar seria reutilizando componentes. Segue-se uma breve reflexão do porquê da não reutilização de componentes e que levou ao trabalho desenvolvido.

A plataforma OutSystems encontra-se bem estruturada em vários componentes. Esta separação foi efectuada tendo em conta a actividade que cada componente é responsável por realizar. Este facto facilita a extensão da plataforma.

Após se proceder à análise dos componentes para compreender quais destes necessitariam de ser adaptados, verificou-se que os componentes directamente ligados à execução eram os candidatos principais a alterações, mais concretamente os componentes internos do *HubServer*.

A solução para a execução de processos passaria sempre pela escolha do Motor de Execução utilizado. Este é o componente interno capaz de receber uma definição de um processo e orquestrar toda a execução deste. É claro para este caso particular o componente de execução teria de ser integrado com a plataforma já existente e dentro do possível manter as características pelas quais esta é conhecida.

O facto de os processos de negócio serem uma área em franca evolução, levou a que ao longo do tempo fossem surgindo vários componentes disponíveis quer para investigação, quer para simples utilização, evitando que quem deles precise necessite de reproduzir todo o trabalho de raiz.

Existem disponíveis livremente, Motores de Execução que poderiam ser usados no caso particular da OutSystems, desde que a estrutura desta permitisse a inclusão de um componente deste tipo.

Visto o *HubServer* apresentar-se bem estruturado por componentes, seria possível integrar todo um novo componente externo que ficasse responsável pela execução de processos. Contudo à que avaliar se seria a melhor opção tendo em conta os pontos tecnológicos e os objectivos a que a OutSystems se propõe. Seguindo este raciocínio segue-se uma breve avaliação dos prós e contras que uma solução deste tipo traria.

Como se pode ver pela Figura 12, esta solução não levanta à primeira vista qualquer tipo de problema estrutural, no entanto seriam necessárias várias adaptações ao componente a incluir e também ao HubServer para que esta solução funcionasse:

- Adaptação ao gerador de código, para que este gerasse o *input* para o Motor de Execução.
- Criar alterações ao Motor de Execução para que este executasse sobre a base de dados da plataforma.
- Criar uma interface com o Motor de Execução para que as aplicações desenvolvidas na plataforma pudessem efectuar alterações aos processos em execução.
- Teria de ser desenvolvida uma nova forma para tratar os erros provenientes da execução de um processo.

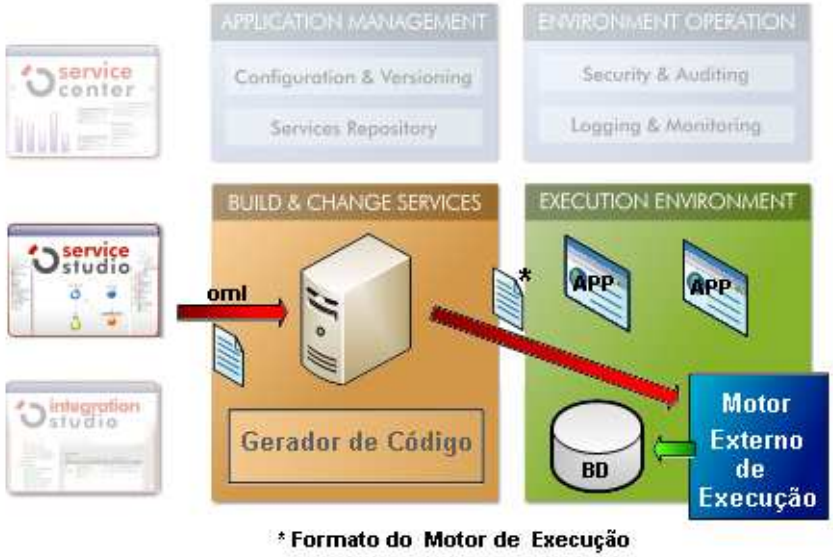


Figura 12 : Estrutura de plataforma com um componente de execução externo

Mostra-se de seguida a uma avaliação das vantagens e desvantagens do uso de um componente externo face ao desenvolvimento de um componente de raiz.

Vantagens	Desvantagens
• A maioria deste tipo de Motores de Execução já suporta BPEL, a norma da indústria para execução de processo de negócio. • Pode trazer mais funcionalidades mais rapidamente.	• São necessárias remodelações fortes ao Motor cuja estrutura interna não é bem conhecida. • A execução de processos iria seguir uma aproximação diferente da execução das aplicações OutSystems. • O facto de se utilizar algo que não foi projectado para as necessidades particulares da OutSystems pode dificultar futuras extensões. • É uma tecnologia externa à OutSystems que pode não proporcionar a performance e fiabilidades promovidas.

Verifica-se que as desvantagens do uso de um componente externo em relação ao desenvolvimento de um componente de raiz superam as vantagens. As alterações necessárias ao Motor seriam vastas, além de que este teria sempre uma arquitectura diferente da restante plataforma, o que poderia levantar problemas com alterações futuras e iria contra um dos objectivos que pretendia manter a estrutura da plataforma o mais inalterada possível.

4. Proposta

Tendo como base a situação levantada no capítulo anterior, trabalhei juntamente com a equipa responsável pela arquitectura da plataforma OutSystems para encontrar uma proposta que endereçasse os problemas existentes e que estivesse alinhada com a evolução que esta plataforma mantém. Esta equipa não só têm um conhecimento profundo da arquitectura como compreendem o que é esperado da plataforma, fazendo com que esta se mantenha alinhada com os objectivos da OutSystems.

Para esta fase do trabalho, ajudou bastante a investigação efectuada sobre as ferramentas e tecnologias actualmente no mercado. Isto permitiu avaliar e compreender o que é esperado de uma ferramenta de execução de processos de negócio e em alguns casos o modo como estas ferramentas funcionam. Toda esta fase ajudou bastante na recolha de ideias para se conseguir chegar a uma proposta estável e alinhada com as necessidades gerais e particulares dos processos de negócio e da OutSystems.

Obviamente a proposta que se segue não foi desenvolvida de uma só vez, tendo sido submetida a várias avaliações e iterações onde se corrigiram erros e se foi adaptando a plataforma considerando sugestões vindas de elementos internos à OutSystems com algum conhecimento sobre a área de processos de negócio.

Segue-se então a descrição da solução proposta. Fazer-se-á uma descrição dos requisitos necessários ao funcionamento da proposta aqui apresentada, em seguida realiza-se uma passagem pela estrutura da plataforma para compreender os componentes que foram adicionados ou alterados, e finalmente descreve-se a arquitectura proposta para os componentes.

4.4 Requisitos

Para a solução de execução que se propõe é necessário que o componente de modelação seja igualmente estendido para permitir a modelação de processos de negócio.

Embora a extensão do componente de modelação não faça parte deste trabalho existem factores que irão afectar a implementação da execução dos processos na plataforma. Um destes factores é a linguagem de modelação usada por este componente. Esta linguagem levanta certos requisitos que terão de ser endereçados na implementação da execução de processos.

Segue-se uma breve descrição da linguagem e das funções de manipulação de processos adicionadas a este componente, que foram desenvolvidas num trabalho paralelo [8].

4.4.1 Linguagem de Modelação

O conjunto de primitivas utilizadas na linguagem do componente de modelação [9] e que posteriormente tiveram de ser levadas em conta no desenvolvimento de toda a estrutura de execução de processos são as apresentadas na Figura 13.

Figura 13 : Elementos da linguagem de modelação

A modelação de um processo dá-se utilizando estas primitivas e setas designadas de transições que ligam os vários nós, dando origem a um *flow* de processo

Tendo em vista a execução temos:

- **Start –** Nó que assinala o ponto de inicio de um processo.
- **Task –** Neste nó pode-se executar lógica de negócio. Antes de se passar ao nó seguinte no *flow* do processo é esperada a chegada de um **Evento**, uma notificação externa que assinala que o processo pode continuar a sua execução
- **AutomaticTask –** Nó onde se executa lógica de negócio, após a qual a execução segue para o nó seguinte do *flow* do processo.
- **SubProcess –** Nó que assinala a execução de uma instância de um processo. O processo responsável pela invocação, espera pela terminação da instância invocada antes de continuar a sua execução
- **GoTo –** É uma abstracção para ligações a outros nós do processo. Do ponto de vista da execução é visto como uma simples transição entre nós.
- **If –** Nó onde se verifica uma condição que pode ser verdadeira ou falsa. Consoante o valor da condição a execução do processo pode seguir caminhos diferentes.
- **Switch –** Neste nó cada transição tem associada a si uma condição. As condições são avaliadas por ordem e quando uma for verdadeira a transição associada é seguida. É necessária uma transição *otherwise* que é seguida caso nenhuma das condições seja verdadeira.
- **Fork –** Nó que permite a criação de fluxos de execução paralelos. Dá-se a divisão de um *Token* da instância em dois ou mais.

- **Join –** Nó que permite sincronizar fluxos de execução paralelos. Cada nó deste tipo tem associada uma área composta por todos os nós entre ele e o menor nó do tipo *fork* que o domina. Após a primeira execução deste nó, só é retomada a execução quando não existirem mais *tokens* na área a ele associada, quer porque já chegaram a este join ou porque saíram da área que este tem associada. Os vários fluxos de execução que chegam a este nó resultam em apenas um fluxo de execução de saída.
- **End –** Nó que assinala um ponto do processo onde o fluxo de execução é terminado.
- **Comment –** Nó que permite inserir comentários no *flow* do processo.

O Motor de execução desenvolvido tem de permitir a execução da semântica associada a cada uma destas primitivas.

4.4.2 Lógica de Negócio

Três dos nós anteriores podem ter associados a si lógica de negócio.

Tal como descrito anteriormente a *AutomaticTask* tem como objectivo executar lógica de negócio, mas para além deste também o nó *Task* e o nó *SubProcess* podem conter lógica de negócio. Para que tal aconteça é permitido que estes definam uma *Preparation* (*flow* já existente na plataforma) que corresponde à lógica de negócio e que é executado antes da execução da semântica do nó.

Este factor levanta a necessidade de comunicação do novo Motor de Execução com a plataforma de execução já existente, pois será recorrendo a esta que a lógica de negócio associada a estes nós será executada.

4.4.3 Funções de Interacção com Processos

Para além das primitivas da linguagem existem ainda outras funções que foram criadas e que têm de ser levadas em conta aquando da implementação

As funções apresentadas na Figura 14, denominadas de *Process API,* representam a forma como as aplicações OutSystems conseguem criar e manipular instâncias de processos.

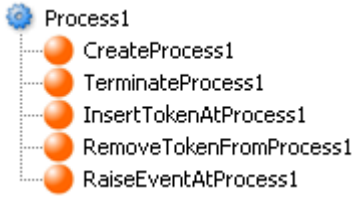


Figura 14 : Funções da Process API

Em detalhe:

- **CreateProcess1** – Criação e execução de uma nova instância de processo que execute a definição de processo *Process1*.
- **TerminateProcess1** – Terminação da instância de processo e de todos os *Tokens* associados a esta. É passado como parâmetro o Identificador da instância a ser terminada.
- **InsertTokenAtProcess1** – Insere um *Token* numa instância da definição de processo *Process1*. A instância e o nó onde o *Token* se irá iniciar são passados como parâmetros
- **RemoveTokenFromProcess1** – Termina um *Token* (fluxo de execução) de uma instância da definição de processo *Process1*. O identificador do *Token* a ser terminado é passado como parâmetro.
- **RaiseEventAtProcess1** – Notifica uma instância da definição de processo *Process1* que pode continuar a executar-se. O identificador da instância é passado como parâmetro. Para esta função ter o funcionamento esperado é necessário que este fluxo de execução esteja parado à espera de uma notificação externa para continuar a sua execução.

Como se pode ver estas funções comunicam directamente com o Motor de Execução pelo que não podem ser esquecidas aquando da implementação deste.

4.1 Abordagem

Uma vez que a opção utilizar componentes externos não preenche completamente as necessidades para este problema, resta-nos analisar quais os componentes que necessitam de alterações e quais terão de ser desenvolvidos de raiz.

Tendo em mente apenas a execução conclui-se que seriam necessárias mudanças a certos componentes internos ao *HubServer* por ser este o responsável pela execução na plataforma. A Figura 15 mostra quais os componentes internos que necessitariam de alterações, dando também uma perspectiva de qual a sua função.

Mais concretamente temos que as alterações necessárias foram:

- O gerador de código teria de gerar a informação necessária para a execução.
- Suporte de base de dados para os processos em execução.
- Todo um novo Motor de Execução para os processos de negócio.

É obvio que para se conseguir uma plataforma completamente funcional e que abrangesse todas as actividades inerentes à Gestão de Processos de Negócio, seriam ainda necessárias mais

alterações, mas para a execução que era o objectivo fulcral deste trabalho, estes eram os componentes que necessitavam de ser endereçados.

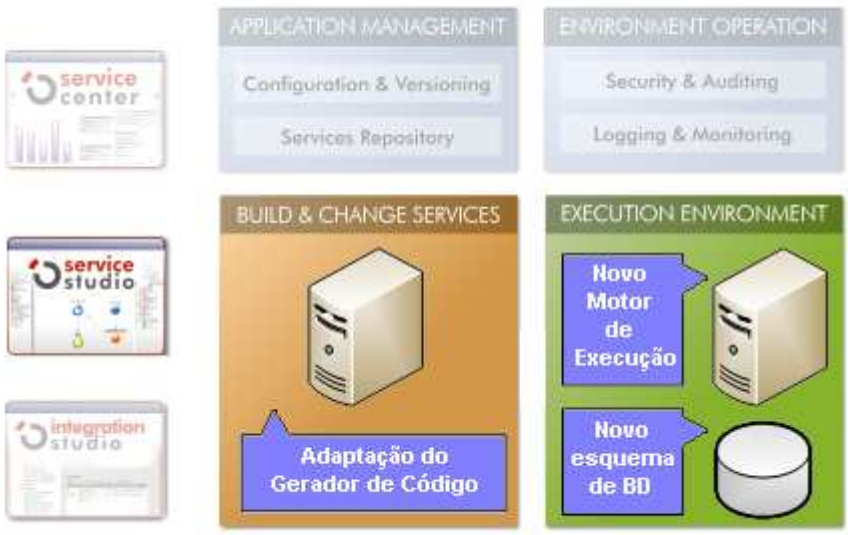


Figura 15 : Adaptações necessárias à plataforma

Na secção seguinte descreve-se os objectivos e arquitecturas para estes três componentes. A ordem por que são listados não está de forma alguma relacionada com a ordem pela qual foram desenvolvidos.

4.2 Arquitectura

Um dos principais objectivos esperados dos componentes seguintes, é que estes apresentem as mesmas características que a restante plataforma apresenta. Devem ser flexíveis o bastante para permitir futuras extensões, e manter a agilidade conhecida. Deste modo, estes aspectos estiveram sempre presentes no decorrer do desenho da solução dos componentes.

4.2.1 Gerador de Código

No contexto das aplicações o gerador de código é responsável por gerar o código da aplicação que foi desenvolvida visualmente no *Service Studio* (componente de desenvolvimento). Embora os processos possam ter lógica de negócio associada, são maioritariamente uma orquestração de actividades e transições entre estas, pelo que neste contexto a função do gerador de código será ligeiramente diferente.

Este ficará responsável pela criação de definições de processos. Após esta criação deverá ser possível criar instâncias de processos desta mesma definição, assim é necessário que após esta fase tanto o Motor de Execução como a base de dados (BD) já conheçam a definição criada.

O gerador de código fica então responsável por interpretar os *ProcessFlows* (definições de processo provenientes do modelador) recebidos no *oml* criado no *Service Studio* e produzir as classes das definições que irão alimentar o Motor de Execução. Para além desta função é também responsável pela população da BD com os dados das definições criadas.

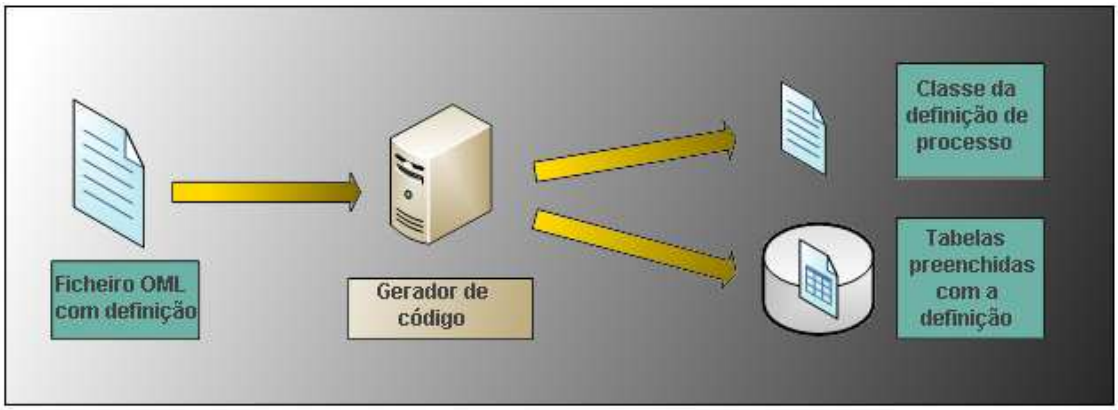


Figura 16 : Inputs e respectivos outputs produzidos durante a geração de código

4.2.2 Motor de Execução

O ambiente *runtime* dos processos vai corresponder a um Motor de Execução desenvolvido de raiz de maneira a endereçar todas as necessidades apresentadas pela plataforma OutSystems. Terá ainda a vantagem de ficar de acordo com a tecnologia da plataforma e respeitar a arquitectura desta.

Este Motor de Execução será o ponto central de toda a execução de processos, sendo que todos os outros componentes deverão ser adaptados às necessidades deste. Este será alimentado pelas definições oriundas do gerador de código e terá uma ligação forte com a base de dados (BD) e com as aplicações em execução na plataforma.

O modelo de execução proposto baseia-se em dois tipos de objectos, as instâncias de processo que representam a execução de uma definição de processo e os *Tokens* que representam um fluxo de execução interno a uma instância de processo. Durante a execução os *Tokens* das instâncias percorrem os vários nós que compõem as definições de processo executando a semântica associada a estes. A presença de um *Token* num nó indica que esse nó se encontra em execução. Casos de paralelismo são representados por instâncias com mais que um *Token*.

Todo o motor foi desenvolvido tendo em conta a manipulação destes objectos. Durante a execução este componente receberá pedidos de manipulação dos processos, actuará sobre os objectos anteriores e guardará na BD o estado das instâncias e *Tokens*.

A arquitectura apresentada na Figura 17, pretende responder ao que foi referido.

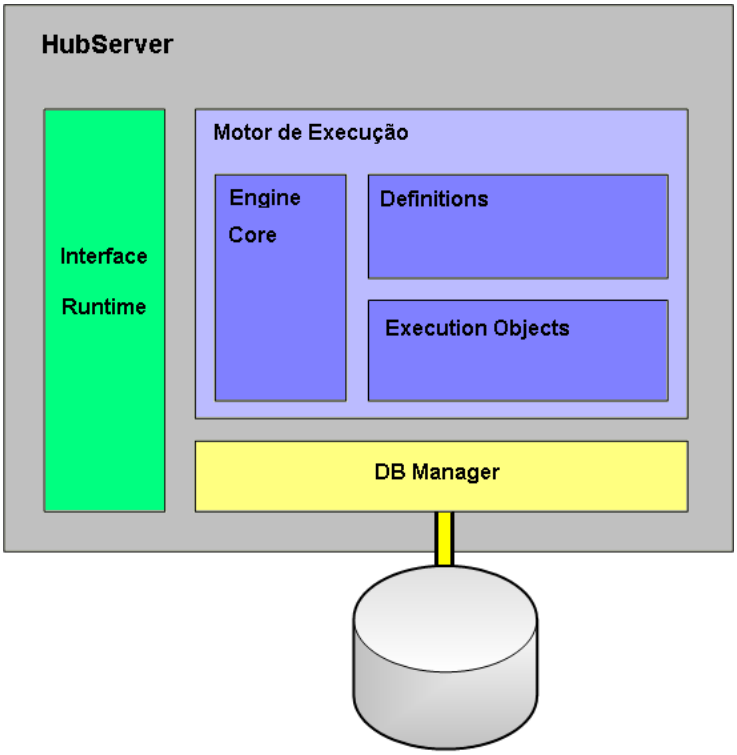


Figura 17 : Arquitectura do Motor de Execução

Segue-se uma breve descrição das funções dos vários elementos da arquitectura do Motor de Execução.

- A **Interface Runtime** fornece um conjunto de funções para manipulação dos processos em execução. Esta oferece funções como criação, eliminação e alteração do estado de instâncias de processos entre outros. O objectivo principal passa por fornecer uma interface bem estruturada para interacção com o Motor de Execução, que escude a implementação interna do exterior e facilite todas as alterações necessárias sem que se necessitem posteriores adaptações aos elementos externos.

- O **DB Manager** é um elemento já existente na plataforma e que foi estendido para o caso particular dos processos de negócio. Este elemento tem como função fazer a gestão das ligações à BD para toda a plataforma, permitindo que se faça uma utilização mais correcta desta. Para o caso particular dos processos tiveram de ser desenvolvidos

métodos de acesso às novas tabelas dos processos para leitura e escrita da informação destes.

- É no elemento **Definitions**, que é criada a estrutura que deve ser seguida pelas definições de processo geradas no gerador de código. Este elemento fornece também todas as funcionalidades gerais das definições que são independentes das definição específicas de cada processo, como por exemplo a semântica associada a cada nó. Isto cria uma separação entre os nós da linguagem e o Motor de Execução, abrindo as portas a futuras alterações à linguagem.

- O elemento **Execution Objects** é onde se dá a execução propriamente dita. O processamento da lógica associada aos processos é efectuado neste elemento, pelo que é também responsável por manter na BD dados sempre actuais e correctos sobre os processos em execução. Este elemento encontra-se fortemente ligado com o anterior pois recorre às definições dos processos e nós definidas nesse objecto.

- O **Engine Core** é o responsável pela gestão de toda a execução. Este trata todos os pedidos recebidos pelo Motor de Execução, começando por verificar a sua validade e garantindo a sincronização entre as instâncias em execução. Posteriormente recorre aos outros elementos do Motor de Execução para a execução dos pedidos recebidos. É responsável pela criação e terminação das instâncias de processos.

4.2.3 Modelo de Dados

Sendo os processos, fluxos de execução que se mantêm por longos períodos de tempo, vai ser através da base de dados (BD) que se suportará a execução *long-running* dos processos. Desta forma a BD vai ser o repositório central de informação sobre os processos, onde estará reunida a informação dos processos não executados, dos executados e dos que ainda se encontram em execução.

Como a BD é o ponto central de informação esta vai ser a base para componentes de monitorização que venham a ser desenvolvidos para a plataforma. Levando isto em conta, deve-se guardar na BD não só a informação exclusiva à execução, mas também outras informações relevantes, como por exemplo o tempo de início e fim de execução dos processos.

De modo a enriquecer a informação sobre os processos, optou-se por incluir na BD toda a definição dos processos. Esta opção surgiu tendo em conta que seria útil ter a informação da definição do processo disponível para a futura componente de monitorização ou para outros casos em que esta fosse útil.

Uma vez que os processos em OutSystems estarão disponíveis para serem manipulados pelas aplicações desta mesma plataforma, a BD deverá também estar disponível para estas aplicações

para que se possa consultar o estado dos processos e agir de acordo. Isto leva que haja um cuidado extra na formatação dos dados presentes na BD, pois estes têm de ser perceptíveis para os *developers* que necessitarem de os consultar.

Resultante dos factores anteriores, temos que a BD vai ser composta por duas partes principais, uma parte estática que corresponderá às definições dos processos e uma parte dinâmica onde serão guardadas as instâncias e todo o tipo de fluxo de execução dos processos. A Figura 18 resume a ideia anterior e explicita que elementos gerais serão guardados em que parte.

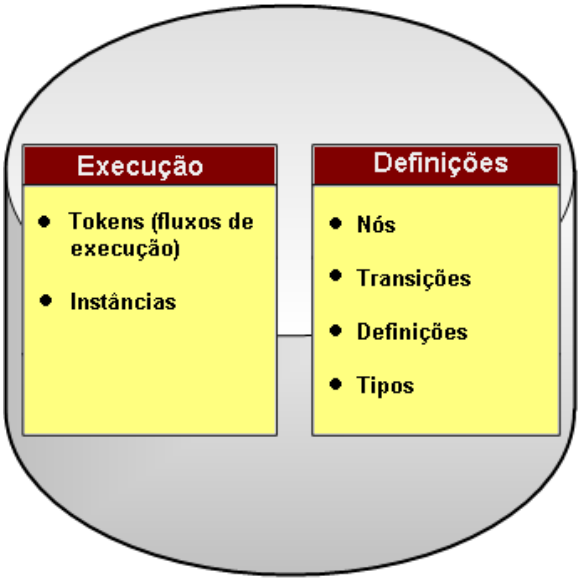


Figura 18 : Estrutura da Base de Dados

4.2.4 Arquitectura Global

Resta apenas compreender como é que esta solução irá funcionar sobre a plataforma OutSystems. A Figura 19 dá um aspecto geral da estrutura do desenvolvimento e execução dos processos na plataforma.

Após o desenvolvimento dos processos no *Service Studio* estes são publicados no *HubServer*, nesta fase é feito o *upload* do ficheiro *oml* com as definições dos processos, após o qual este é usado pelo gerador de código para produzir os ficheiros com as definições dos processos que vão alimentar o Motor de Execução e para carregar as tabelas na base de dados (BD) com as definições dos processos. O desenvolvimento termina neste ponto.

Em tempo de execução chegam pedidos à plataforma cuja lógica leva à alteração do estado de um determinado processo. O Motor de Execução é notificado através da sua interface, processa o pedido actuando sobre a BD, fazendo avançar o estado do processo ou criando novas instâncias de processos, executando sempre a lógica que o processo tenha associada.

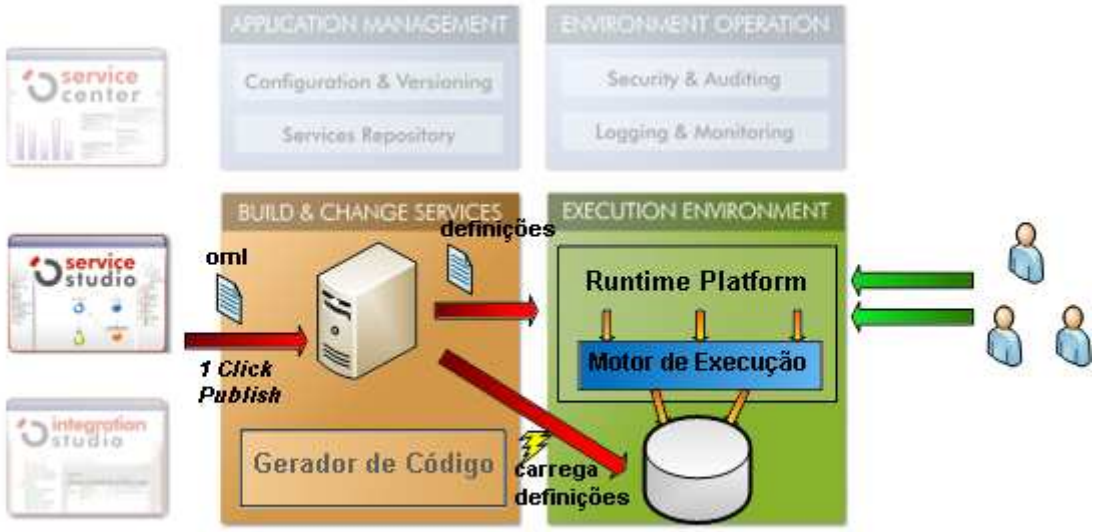


Figura 19 : Arquitectura Global

Consegue-se desta forma executar processos sem alterações profundas à estrutura da plataforma, mantendo o método de desenvolvimento e outras características que proporcionam a flexibilidade pela qual esta plataforma é conhecida.

5. Implementação

Neste capítulo detalha-se a implementação da solução proposta no capítulo anterior.

A implementação decorreu em várias fases, quer pela necessidade de experimentar implementações alternativas, quer por ser necessário abordar vários componentes. Apesar disto, este capítulo detalha a implementação final, explicando as opções tomadas em vez de abordar todas as fases que nada iriam acrescentar.

Inicia-se com uma breve introdução à metodologia de desenvolvimento seguida no desenvolvimento desta solução. Em seguida passa-se a cada um dos componentes desenvolvidos, começando pelo gerador de código, motor de execução, base de dados e finalmente o contexto dos processos.

5.1 Metodologia

Durante o desenvolvimento de todo este trabalho foi seguida a metodologia ágil promovida pela OutSystems, já descrita anteriormente.

Assim sendo foram realizadas as seguintes práticas:

1. Iterações de duas semanas, nas quais eram estabelecidos os objectivos para as duas semanas seguintes.
2. Reuniões de pé, diárias, com a duração de poucos minutos onde era efectuado o relato dos progressos bem como dificuldades encontradas e decisões tomadas.
3. Sempre que possível, desenvolveram-se primeiro, as tarefas que trouxessem mais beneficio para a solução.

A implementação aqui apresentada surgiu depois de vários níveis de prototipagem que permitiram estudar as características da solução com os objectivos concretos que se tinham definido, para tal foram efectuados vários testes recorrendo a processos tipo que explorassem as várias potencialidades da solução do ponto de vista da execução

5.2 Gerador de Código

O gerador de código é um caso particular no que diz respeito à sua implementação, pois este possuía já uma estrutura bem definida que não deveria ser alterada, mas sim estendida para suportar a geração necessária aos novos *flows* e funções acrescentadas ao modelador. Neste sentido deve-se começar por analisar as alterações ao *input* recebido no gerador.

Como já foi visto anteriormente o gerador de código recebe um ficheiro *oml* como *input*,que não é mais que um ficheiro xml protegido. Com a inserção de novos *flows* na plataforma o xml produzido

pelo modelador passou a incluir novos elementos. A Figura 20 mostra exemplos de elementos do xml produzido que foram incluídos ou que sofreram alterações.

Figura 20 : Novos elementos da OML

Estes elementos necessitaram de ser agora endereçados pelo gerador de código pois é nestes que se situa toda a informação dos processos e das funções que irão permitir uma forte interacção entre as aplicações e processos em execução. É esta informação que terá de ser interpretada pelo gerador de código para produzir os resultados esperados.

O processamento dos ficheiros xml no gerador de código é realizado em duas fases, a criação de um modelo em memória e a geração do output propriamente dito que utiliza o modelo previamente criado. Surgiu ainda um caso de excepção para o qual o processamento realizado teve de ser realizado de maneira ligeiramente diferente.

5.2.1 Criação do Modelo

O gerador continha já um largo conjunto de objectos para os elementos já existentes que teve de ser alargado para permitir a criação do modelo tendo em conta os elementos dos processos de negócio.

Neste sentido, para permitir o processamento dos *ProcessFlow* foram criados os objectos de acordo com a Figura 21.

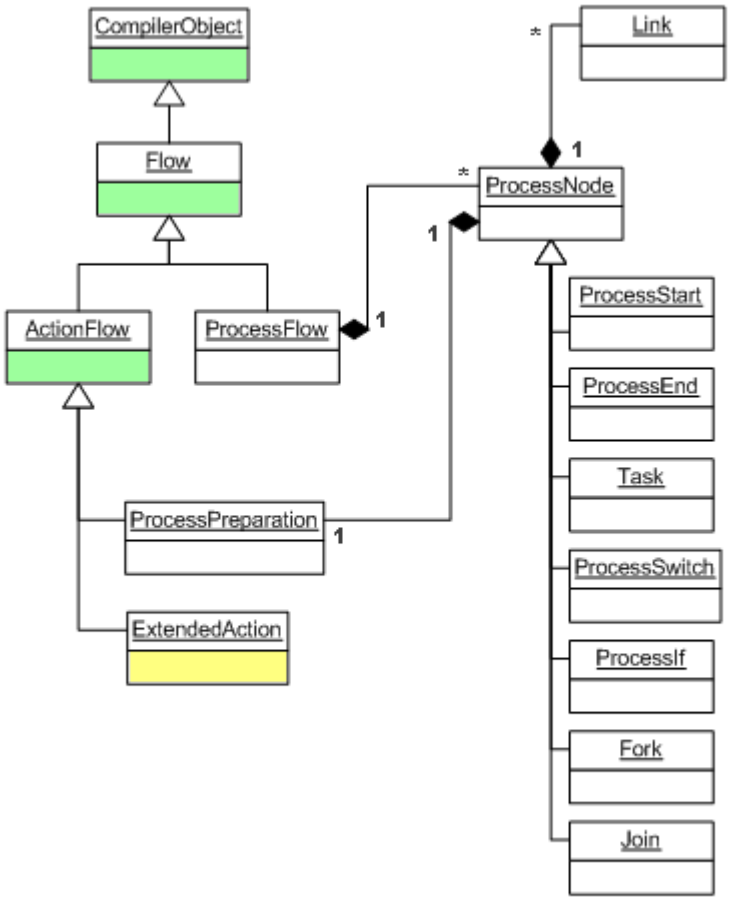


Figura 21 : Diagrama de classes (A verde as classes já existentes, a amarelo as classes que necessitaram de alterações, a branco as classes criadas)

Em detalhe:

- **ProcessFlow –** Objecto referente ao flow de um processo.
- **ProcessPreparation –** Objecto referente ao flow da lógica de negócio que pode ser definido para alguns dos nós do flow processo.
- **ProcessNode –** Objecto referente aos variados nós utilizados nos flows de processo. Este é um objecto geral pelo que foi criada uma especialização deste objecto para cada um dos nós disponíveis no flow de processo.
- **Link –** Objecto referente às transições existentes entre os nós do flow de processo.
- **ExtendedAction –** Este objecto particular já existia, contudo teve de ser estendido para permitir endereçar as funções de interacção com o ProcessEngine.

Com esta nova extensão passou a poder criar-se um modelo em memória que englobasse as definições de processos e funções relacionadas com estes.

Nesta implementação teve-se ainda o cuidado de dar flexibilidade a futuras alterações à linguagem, desta forma caso seja necessária a inclusão de novos nós na linguagem, apenas é necessário criar uma especialização do objecto *ProcessNode* para as particularidades do novo nó.

5.2.2 Geração do Output

A finalização da criação do modelo dos vários flows em memória significa que a definição não continha erros e consequentemente está criada a base para a geração dos outputs.

O passo seguinte é o preenchimento da base de dados (BD) com as novas definições de processos. Este passo tem de ocorrer obrigatoriamente antes da geração das classes de definições de processos. Isto deve-se ao facto dos identificadores das definições serem produzidos pela BD, o que permite centralizar a gestão dos identificadores e possibilita a existência de mais que um gerador de código a produzir definições para a mesma BD.

No respeitante ao preenchimento da BD existe uma separação entre os objectos referidos anteriormente:

- **Definição de processos –** Os objectos que se referem à definição de processos propriamente dita, necessitam de ser carregados na BD, pois são necessários para especificar a definição de processos e o estado dos processos em execução. Os objectos nesta situação são o ProcessFlow, todos os ProcessNodes e os Links.
- **Lógica de negócio –** Os restantes objectos, nomeadamente o ProcessPreparationFlow e as ExtendedActions referem-se apenas a lógica de negócio, pelo que não necessitam de carregar qualquer informação na BD, apenas interessa o código a que estes derem origem.

Neste sentido definiram-se para os objectos referentes à definição de processos funções ***DumpDataOnBD*** responsáveis pela população das respectivas tabelas na BD e pela recolha dos Identificadores atribuídos.

Além disto, para todos os novos objectos foram definidas funções ***DumpCode*** responsáveis por gerar o código propriamente dito. Para os objectos referentes às definições de processos, temos que um *ProcessFlow* vai dar origem a uma classe que contem os nós, as ligações existentes entre eles, os métodos para a instanciação da definição e os métodos de invocação da lógica de negócio associada a esta. Os *ProcessNodes* vão ser agregados na classe gerada pelo *ProcessFlow*. Para os restantes objectos referentes à lógica de negócio, o código gerado é em tudo semelhante ao que já era gerado para as aplicações, pelo que se recorreu a funções já existentes.

Desta forma após a criação do modelo da aplicação é invocada a função responsável pela geração de código, que vai realizar uma invocação recursiva aos objectos do modelo para geração do output necessário.

No final deste processamento a BD já se encontra preenchida com as novas definições e os ficheiros já se encontram gerados. Agora a aplicação será compilada e instalada no ambiente de execução pelo *DeploymentController* da plataforma, ficando a definição pronta para execução.

5.2.3 Casos Particulares

Como referido anteriormente o processo de geração é iniciado pela criação de um modelo dos nós do processo em memória, o qual é utilizado para gerar o código respectivo.

Para a maioria dos nós a criação deste modelo é bastante simples, não sendo necessário mais que a criação de um objecto referente ao nó, no entanto para o caso do nó *join* esta situação torna-se um pouco mais complexa devido às particularidades da semântica deste.

Tal como descrito no capítulo anterior, os *joins* têm associado um conjunto de nós, designado área de influência do *join* que irá afectar o modo como este se comporta em tempo de execução.

Sendo que esta área apenas é necessária em tempo de execução, poderia calcular-se nessa altura, contudo essa implementação aumentaria a carga depositada no Motor de Execução pois cada instância teria de fazer esse cálculo. Visto que a área dos *joins* é estática, ou seja, não se altera durante a execução, a melhor solução será aproveitar o pré-processamento já efectuado durante a geração de código para proceder ao cálculo dessa área.

Desta forma nas classes de definições geradas pelo gerador de código já estão descriminados qual os nós que fazem parte da área de influência de cada *join* dessa definição. Isto leva a que o cálculo desta área seja efectuado apenas uma vez para cada *join* de cada definição, aquando da criação do objecto no gerador referente ao nó *join*.

Na altura da criação do modelo do *flow* este é pré-processado para ser possível o cálculo das áreas dos *joins*. Obviamente este pré-processamento só ocorre caso o *flow* inclua nós de *join*. Recorrendo a funções de manipulação de *flows* que baseando-se no pré-processamento anterior permitem calcular para um nó quais os seus nós dominadores e atingíveis, chega-se facilmente à área do *join* pelo seguinte procedimento:

1. Cálculo do menor dos *fork's* dominadores do nó *join*.
2. Caso não existam *fork's* dominadores é dado um erro.
3. Cálculo do conjunto de nós atingíveis a partir do menor *fork* dominador
4. Cálculo do conjunto de nós que atingem o nó *join*.
5. Cruzamento dos dois conjuntos anteriores.

Na altura de geração do código do *join*, este procedimento é executado e o conjunto de nós resultante é usado para gerar a área associada àquele *join* particular. Este procedimento é executado para todos os *joins* presentes no *flow*.

A Figura 22 apresenta um exemplo de um flow com a área do nó sobressaída.

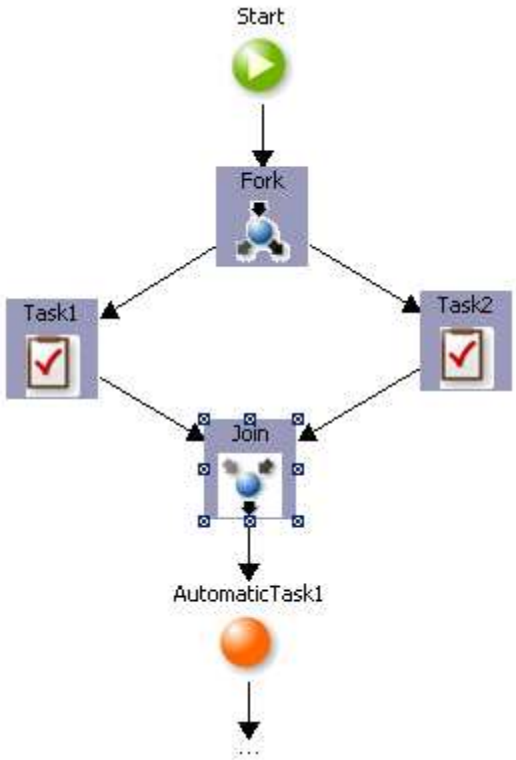


Figura 22 : Exemplo da área de influência do nó Join

5.3 Motor de Execução

O Motor de Execução é o elemento central desta solução. Este foi desenvolvido de raiz e integrado na plataforma existente, o que permitiu que as aplicações existentes possam assentar a evolução da execução num processos de negócio, ou possam agir sobre os processos de negócio que já se encontram em execução.

Segue-se uma breve descrição detalhada do funcionamento interno deste motor de execução para descrever as suas funcionalidades, particularidades e para se tornar mais simples a compreensão da implementação deste motor.

5.3.1 Descrição

A execução de um processo é feita em torno de vários objectos, que vão evoluindo ao longo do tempo consoante os pedidos que são processados sobre estes.

Para se perceber este processamento é importante distinguir entre os seguintes conceitos:

- **Definição de processo –** Conjunto de nós e ligações que descrevem o fluxo e as actividades de um processo de negócio.

- **Instância de processo –** Objecto que representa a execução de uma definição de processo.
- ***Token* de processo –** Objecto que corresponde a um fluxo de execução.

Pelas definições anteriores temos que uma definição de processo pode dar origem a várias instâncias de processo, uma por cada vez que a execução da definição de processo é iniciada.

Por sua vez todas as instâncias de processo têm pelo menos um fluxo de execução, o que corresponde a dizer que contêm pelo menos um *Token*. Para os casos em que as definições de processo descrevem processos com casos de execução paralela, estas dão origem a instâncias de processo com mais do que um *Token*, um para cada fluxo de execução presente.

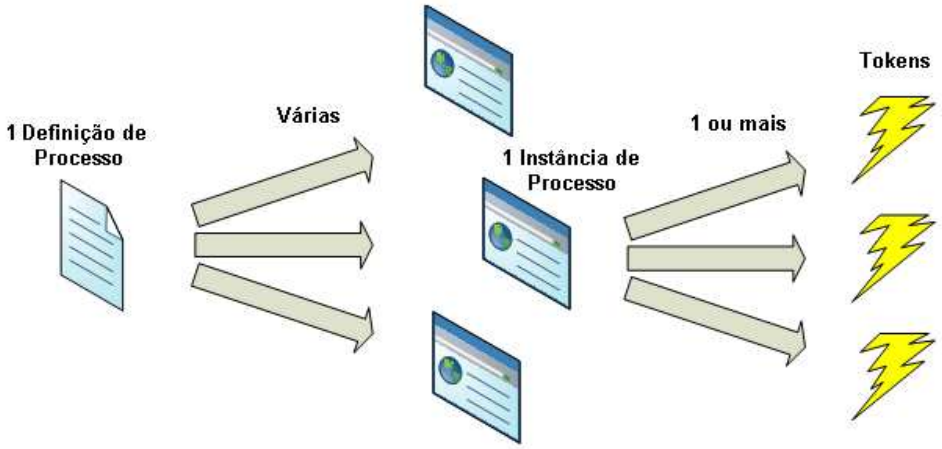


Figura 23 : Definições, Instâncias e *Tokens*

Embora sejam as instâncias de processo que marquem a execução de uma definição, a execução propriamente dita é marcada pela passagem dos *Tokens* pelos vários nós da definição de processo. A chegada de um *Token* a um nó assinala que esse nó se encontra em execução.

Durante a execução e à medida que os *Tokens* vão avançando nos processos, estes podem encontrar-se em vários estados consoante o que está a ser executado de momento. A lista seguinte mostra os estados possíveis de um *Token*.

- **Running –** A executar o processo e a lógica associada a este.
- **WaitEvent –** Parado à espera de um evento externo para que possa retomar a execução.
- **WaitSubProcess –** Parado à espera que a execução de um sub-processo iniciado por este token termine e o notifique para que possa retomar a execução.
- **Finnished –** O *Token* em causa terminou a sua execução.

- **WaitSync –** *Token* que espera uma sincronização para iniciar a sua execução (sub-secção: Casos Particulares).

Por sua vez as instâncias têm apenas dois estados, ou estão em execução ou estão terminadas caso em que todos os seus *Tokens* estejam no estado “Finnish” e lhe tenha sido atribuída uma data de terminação.

As instâncias de processo assim como os *tokens* residem na base de dados (BD) sendo apenas reconstruídos em memória quando é necessário executar alguma operação sobre eles. As definições de processo, por sua vez, existem em memória e na BD, pois para além de guardarem os dados da definição guardam também a lógica associada a cada nó e a lógica particular de cada processo, oriunda do gerador de código e que não pode ser guardada na BD.

Após compreendermos os principais elementos criados para a execução de processos, iremos agora abordar como é feito o processamento de pedidos no Motor de Execução.

5.3.2 Tratamento de Pedidos

É natural encontrar processos de negócio que se executem por longos períodos de tempo, assim optou-se por um motor que funcionasse através do processamento de pedidos, ou seja, a execução no motor é desencadeada pela chegada e processamento de pedidos ao Motor de Execução. Para esse efeito este disponibiliza uma interface para o exterior, denominada de *Interface Runtime* que é composta pelos vários pedidos que o Motor é capaz de processar. Os elementos da *Process API* disponíveis no modelador utilizam os métodos da *Interface Runtime* disponibilizados pelo Morto de Execução.

A *Interface Runtime* disponibiliza para o exterior os seguintes métodos:

- **CreateProcess –** Criação de uma nova instância de Processo.
- **StartProcess –** Inicia a execução de uma instância de processos previamente criada. A separação deste método do anterior foi necessária para o preenchimento do contexto de uma instância de processo antes de esta iniciar a sua execução, mas após a sua criação.
- **CreateSubProcess –** Cria e inicia uma instância de processo. Esta instância criada corresponde a um sub-processo, ou seja quando terminar terá de notificar a instância criadora para que esta retome a execução.
- **InsertToken –** Insere e inicia um *token* numa instância de processo já existente. É necessária a indicação do nó em que o *token* deve ser criado.
- **RaiseEventAtToken –** Envia um evento para um *token* específico. Caso o *token* esteja à espera do evento a execução é retomada, caso contrário o evento é ignorado.

- **RaiseEventAtInstance –** Envia um evento para a instância. Caso existam *tokens* da instância à espera deste evento a sua execução é retomada, caso contrário o evento é ignorado.
- **RemoveToken –** Termina a execução do *token*. Coloca o *token* no estado *"Finish"*.
- **DeleteInstance –** Termina a execução da instância de processo. Coloca todos os *tokens* da instância no estado *"Finish"* e define a data de terminação da instância.

Todos os pedidos recebidos são processados assincronamente. Após a recepção de um pedido é verificada a validade deste e é criada uma *Thread* que o irá processar libertando rapidamente o sistema para o atendimento de novos pedidos. Este tipo de processamento aumenta bastante a escalabilidade do motor de execução pois desta forma passa a ser possível o tratamento de vários pedidos paralelamente, no entanto levanta um problema de sincronização, pois passa a ser possível a execução paralela de vários pedidos sobre o mesmo *Token* de processo, o que não deveria ocorrer.

Para resolver esta nova situação e visto que todos os objectos residem na BD, cada vez que é necessário fazer uma operação sobre um objecto este tem de ser lido da BD e nessa altura é adquirido um trinco sobre esse mesmo objecto que só será libertado quando o processamento sobre esse objecto terminar.

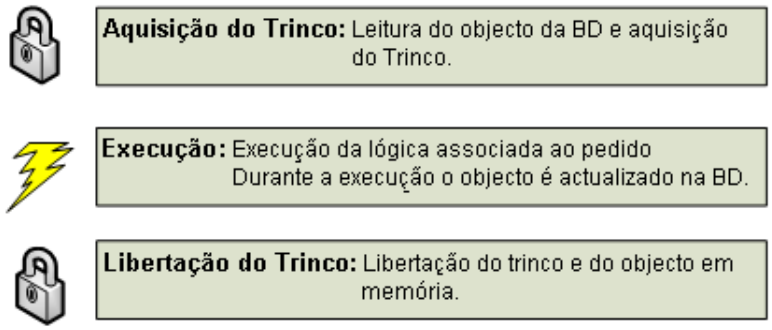


Figura 24 : Processamento de um pedido

A execução do *Token* surge pela chegada de um pedido que no decorrer do seu processamento inicia o *Token* que trata de executar a lógica dos nós da definição que lhe está associada, indo sempre actualizando o seu nó actual e o seu estado.

Durante a execução os nós necessitam de interactuar com o *Token* que os está a executar, quer pela presença de um nó terminal que marca a terminação do *Token*, quer porque o *Token* deve parar a sua execução em espera de um evento ou mesmo outros casos. Para resolver isto os *Tokens* oferecem uma Interface *IToken* que permite aos nós influenciarem a execução dos *Tokens*.

Elementos da Interface *IToken*:

- **SetToKill –** Marca o Token para ser terminado (utilizado maioritariamente por nós *End*).
- **SetWaitEvent –** Marca o *Token* para esperar por um evento no final deste processamento (utilizado maioritariamente por nós *Task* com eventos de saída).
- **SetNextTransition –** Marca qual é a próxima transição a ser seguida pelo *Token* (utilizado maioritariamente por nós *If* e *Switch*).
- **SetNewChildren –** Cria novos Tokens. (utilizado maioritariamente por nós *Fork*).

O caso anterior também é verificado para as instâncias, pois existe a necessidade de nós e *Tokens* comunicarem com estas quando surgem casos de sincronização da execução de processos. O caso típico é a execução de um nó *Join*. As sincronizações são realizadas ao nível da instância pois afectam os vários *Tokens* que a compõem.

Elementos da Interface *IProcessInstance*:

- **CheckArea –** Verifica a existência de Tokens na área associada ao nó (utilizado maioritariamente pelo nó *Join*).
- **Synchronize –** Cria, caso não exista, um ponto de sincronização. Corresponde a criar um *Token* no estado *WaitSync* (utilizado maioritariamente pelo nó *Join*).

As funções anteriores servem para que os nós possam ter semânticas que dependam de vários *Tokens* e não apenas daquele que os executa. O caso particular encontrado na linguagem utilizada é o nó *Join*.

Regra geral a execução de um *Token* tem a estrutura apresentada na Figura 25:

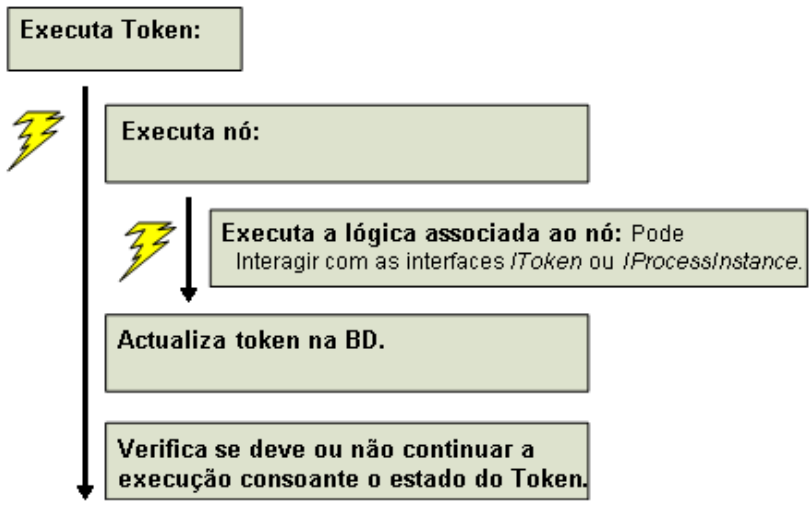


Figura 25 : Execução do *Token*

Os *Tokens* invocam a execução da lógica dos nós recorrendo à interface *INode*.

Elementos da interface *INode*:

- **ID –** Permite aceder ao identificador do nó.
- **Execute –** Permite invocar a execução da lógica associada ao nó.

Recorrendo ao conjunto de interfaces anteriores o Motor de Execução mantém a independência entre os seus componentes internos. Futuras extensões aos componentes são facilmente atingidas pois os componentes comunicam entre si através de interfaces bem definidas ignorando a implementação interna.

5.3.3 Diagrama de Classes

As funcionalidades descritas anteriormente foram atingidas através dos objectos da figura seguinte:

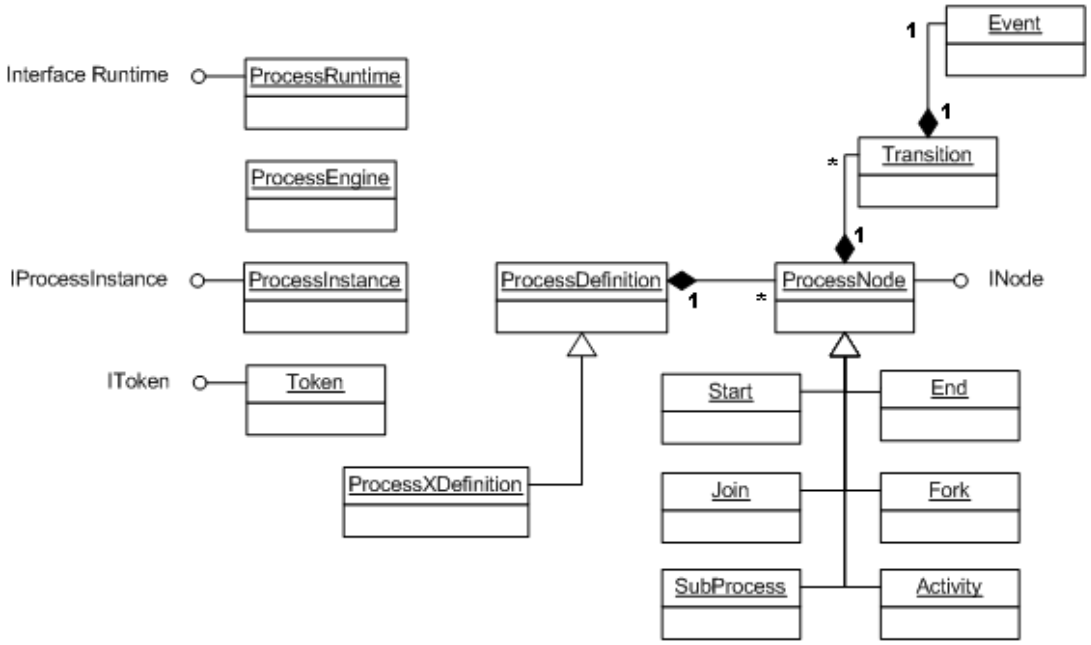


Figura 26 : Diagrama de classes do Motor de Execução

Em detalhe:

- **ProcessRuntime –** Classe que disponibiliza a interface para a restante plataforma comunicar com o Motor de Execução. Os métodos desta classe na sua maioria apenas fazem tratamento dos parâmetros de entrada, e o lançamento de *Threads* para o processamento dos pedidos.

- **Interface Runtime –** Conjunto de métodos que o Motor de Execução disponibiliza para o exterior. Normalmente estes métodos serão chamados por outras aplicações em execução na plataforma.
- **ProcessEngine –** Classe central ao motor de execução. É aqui que se faz todo o processamento dos pedidos recebidos, bem como toda a gestão de trincos necessária ao correcto funcionamento do motor.
- **ProcessDefinition –** Classe geral que implementa os métodos comuns a todas as definições de processo e define a estrutura das classes geradas
- **ProcessNode –** Classe geral que implementa os métodos comuns aos vários nós da linguagem de modelação.
- **ProcessXDefinition –** Classes geradas no gerador de código que correspondem às definições de processo. O *X* no nome desta classe pretende exemplificar o nome do processo, assim o processo *Process1* daria origem à classe *ProcessProcess1Definition*.
- **ProcessInstance –** Classe que implementa uma instancia de execução de um processo de negócio. Devido ao facto das instâncias desta classe comutarem muitas vezes entre memória e base de dados, esta não guarda referências para outros objectos mas sim os identificadores desses objectos, um exemplo é o identificador da definição a executar.
- **Token –** Classe que implementa um fluxo execução de uma instância da execução de um processo de negócio. Tal como a classe anterior o facto das instâncias desta classe comutarem muitas vezes entre memória e base de dados, levou a que esta não guardasse referências para outros objectos mas sim os identificadores desses objectos, um exemplo é o identificador da *ProcessInstance* a que pertence.
- **IToken –** Interface implementada por todos os *Tokens* que disponibiliza métodos que permitem aos nós afectar os tokens que os estão a executar.
- **IProcessInstance –** Interface implementada pelas *ProcessInstances* que disponibiliza métodos para que os *Tokens* e os nós possam afectar a *ProcessInstance* a que pertencem.
- **INode –** Interface que todos os nós implementam através da qual os *Tokens* invocam a execução dos nós da definição de processo que estão a executar.
- **Start, End, Join, Fork, SubProcess –** Estas classes são extensões da classe ProcessNode onde é implementada a lógica particular associada a cada um dos nós da linguagem.
- **Activity –** Esta classe, tal como as classes do ponto anterior, implementa lógica particular aos nós de *If*, *Switch*, *AutomaticTask* e *Task* da linguagem. Optou-se por usar uma mesma classe para o processamento destes nós, porque a principal diferença entre eles deve-se a propriedades que são associadas durante a modelação e não ao seu funcionamento em execução. Neste sentido durante a geração de código estes nós serão processados de forma diferente originando o comportamento diferente que esta primitiva irá produzir.

Este modelo é compatível com a introdução de novos nós à linguagem de modelação. Para tal apenas seria necessária desenvolver uma nova classe que implemente a Interface *INode* e estenda a classe *ProcessNode*. Esta classe deve implementar a semântica referente à nova primitiva. Se for necessário manipular o *Token* ou a *ProcessInstance* apenas é necessário recorrer às interfaces disponíveis.

5.3.4 Casos Particulares

Em tempo de execução, o nó *join* é também um caso particular.

Como dito anteriormente este nó permite sincronizar fluxos de execução (*Tokens*). Após a chegada de um *Token* a um nó *join* que o executa, a continuação da execução do *flow* a partir deste nó só é retomada quando não existirem outros *Tokens* na área de influencia deste. A inexistência de nós na área pode surgir de três formas diferentes:

1. Aquando da chegada do primeiro *Token* ao *join* não existem mais nós na área e a execução pode ser retomada.
2. Todos os *Tokens* presentes na área chegaram ao *join* e o fluxo de execução resultante pode iniciar a sua execução
3. Os *Tokens* presentes na área seguiram caminhos que os levaram a abandonar a área de influência do *join* sem passarem por este.

Os dois primeiros pontos não levantam qualquer problema, pois são resolvidos pela verificação da existência de *Tokens* na área do *join* sempre que um *Token* chegue a este nó, caso não existam *Tokens* na área a execução é retomada.

O terceiro caso levanta um problema que não é resolvido por esta solução. Dado que um nó só se executa quando um *Token* chega até si, a movimentação de um *Token* para fora da área de influência não é notada pelo *join* que está a proceder a uma sincronização.

A solução encontrada passa pela criação de um novo *Token* sempre que o primeiro *Token* executa o *join* e pela terminação dos *Tokens* de entrada no *join*. Este novo *Token* terá um estado diferente dos apresentados anteriormente, o estado *WaitSync* que assinala que se está a proceder a uma sincronização no *join* que é o nó corrente desse mesmo *Token*. Sempre que um *Token* avança no *flow*, isto é, sempre que um *Token* muda o seu nó actual, é feita a verificação de todos os *Tokens* da instância no estado *WaitSync* para verificar se existem ou não *Tokens* na área do *join* correspondente, caso não existam o *Token* é mudado para o estado *Running* e a execução é retomada. A verificação terá de garantir que não existem avanços de *Tokens* enquanto estas estão a ser efectuadas.

Esta solução não só permite resolver o caso anterior, como deixa aberta a possibilidade de novas primitivas da linguagem efectuarem sincronização, pois teriam apenas de proceder da mesma forma, criando um *Token* no estado *WaitSync* e terminando os *Tokens* que entrassem no nó.

5.4 Modelo de Dados

Como referido no capítulo 4, o modelo da base de dados (BD) é composto por duas partes referentes às definições de processos e à execução de processos.

A própria plataforma OutSystems assenta num poderoso esquema de base de dados para fazer a gestão das aplicações. A BD mantém um registo das aplicações em execução, das que foram instaladas e muitas outras informações úteis internamente para a plataforma ou para uso externo no *Service Center* (componente de monitorização e gestão de aplicações).

Este factor teve de ser endereçado no desenvolvimento do modelo de dados, pois este deve integrar-se com o modelo já existente para que a gestão dos processos de negócio possa ser inserida na gestão do *eSpace* (aplicação em OutSystems) que lhe deu origem.

Tendo os factores anteriores em mente chegou-se ao modelo de dados apresentado na Figura 27.

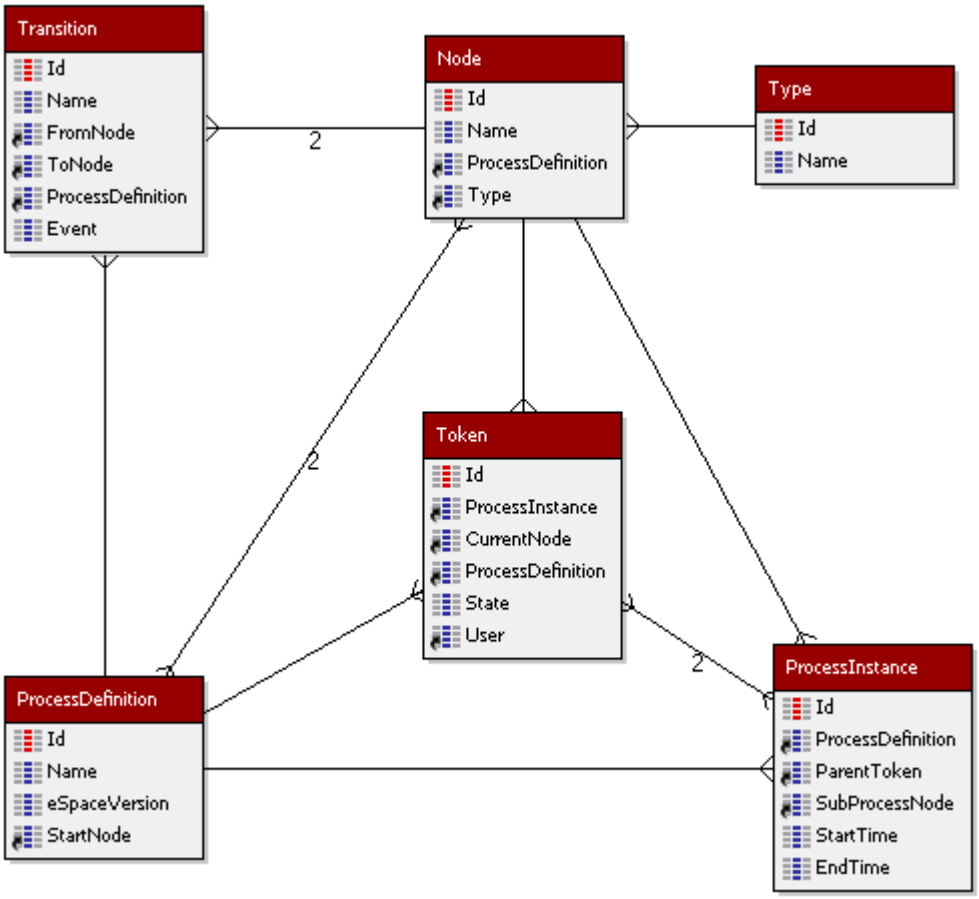


Figura 27 : Modelo de dados

Em detalhe:

- **Tabela ProcessDefinition –** Responsável pelo registo das definições de processo criadas. Guarda-se o nome e uma referência para o primeiro nó que irá ser executado no processo. O atributo *eSpaceVersion* faz a ligação com o restante modelo associando esta definição a uma versão da aplicação em desenvolvimento.
- **Tabela Node –** A tabela *Node* corresponde aos nós que compõem as definições. Embora as definições usem nós com tipos iguais, dois nós do mesmo tipo irão ser representados como duas entradas nesta tabela. Embora este modo exija o armazenamento de mais informação, é necessário pois os nós do mesmo tipo podem possuir nomes diferentes que devem ser preservados para a reconstrução da definição.
- **Tabela Transition –** Esta tabela guarda os links entre os nós pertencentes a uma definição de processo. Os eventos associados às transições são guardados atribuindo um nome ao link, que deverá corresponder ao nome do evento e a marcar o link como tendo um evento associado no atributo *Event*.

  - **Tabela Type –** Esta tabela é geral a todas as definições de processos. É preenchida aquando da instalação da plataforma, com os vários tipos de nós da linguagem de modelação, não voltando os seus dados a ser alterados a não ser que surjam alterações à linguagem de modelação. Esta solução limita erros e facilita as alterações à linguagem caso estas sejam necessárias, pois deste modo a informação sobre os tipos da linguagem encontra-se centralizada num único ponto ao invés de dispersa e replicada por toda a BD.

    Esta tabela pode, de futuro, ser também utilizada para guardar outros tipos associados aos processos.
- **Tabela ProcessInstance –** Esta tabela guarda a informação sobre as instâncias de processo. São guardadas para todas as instâncias a definição que estão a executar e o tempo de início e fim da execução.

  Os atributos *ParentToken* e *SubProcessNode* servem para os casos particulares em que a instância em execução corresponde a um sub-processo. Um sub-processo tem origem quando um fluxo de execução de uma instância (*token*) dá início a uma nova instância e se bloqueia à espera da terminação desta última (ver Figura 28). O atributo *ParentToken* serve para sabermos qual *token* notificar quando a instância correspondente ao sub-processo terminar. O atributo *SubProcessNode* serve para validar a notificação.
- **Tabela Token –** Como referido anteriormente, uma instância corresponde a um ou mais fluxos de execução (*tokens*), a tabela Token guarda a informação sobre os vários fluxos de execução. Para cada fluxo de execução é guardada a instância a que este fluxo pertence (*ProcessInstance*), a definição que este está a executar (*ProcessDefinition*), o nó actual do fluxo de execução (*CurrentNode*), o estado do fluxo de execução (*State*) e o utilizador, caso exista, que está a trabalhar sobre este fluxo de execução.

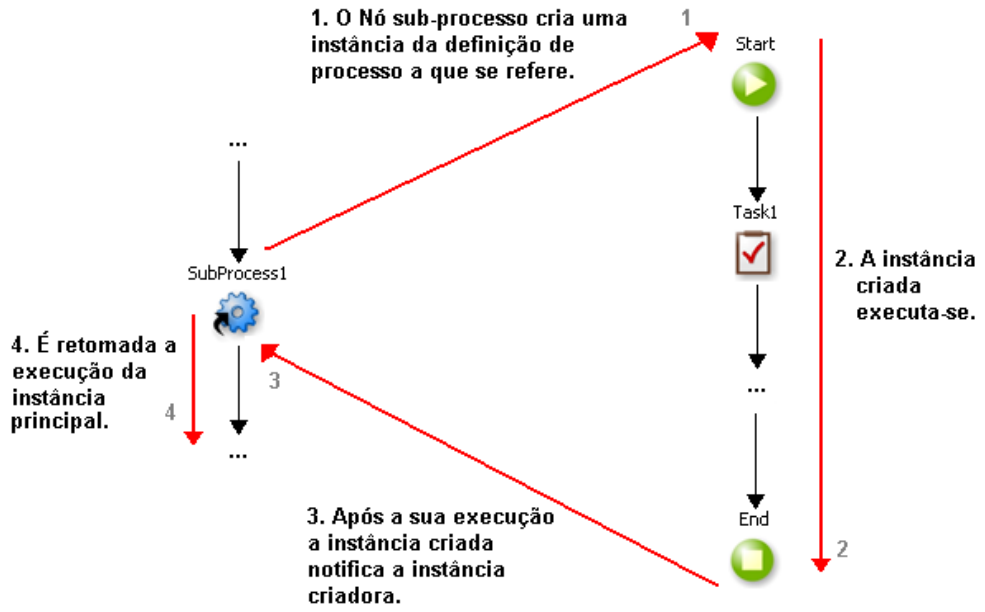


Figura 28 : Execução de um sub-processo

Pela observação do modelo de dados na Figura 27, repara-se que praticamente todas as tabelas possuem uma ligação à tabela *ProcessDefinition*. Este facto cria uma redundância de informação na BD, mas minimiza a carga das acções sobre a BD durante a execução. A concorrência no acesso à base de dados pode ser bastante elevado, dependendo das particularidades das definições de processo e do número de instâncias em execução. A duplicação deste atributo diminui o número de *joins* de tabelas necessários para responder aos acessos à BD, diminuindo assim carga sobre esta.

Apesar de simples este modelo oferece todo o suporte necessário para a execução de processos no Motor de Execução apresentado na secção anterior.

5.5 Contexto

Até este momento não foi referida como é gerida a informação que os processos possam ter associada.

O modelador desenvolvido permite a definição de variáveis de processo e parâmetros de *input* que devem ser fornecidos aquando da criação das instâncias. Ambos os parâmetros e as variáveis estão disponíveis durante toda a execução do processo.

Para guardar estas variáveis é criada para cada definição de processo uma tabela específica onde vão ser guardados os valores destas variáveis e parâmetros para cada instância desta definição

de processo. A tabela criada terá uma coluna com o identificador da instância de processo, para se saber a que instância de processo pertence aquele conjunto de variáveis e ainda uma coluna por cada variável ou parâmetro do processo. A criação da tabela na base de dados dá-se durante o processamento efectuado pelo gerador de código.

Quando uma instância de processo é criada recorrendo à função *CreateProcessX* disponível no modelador, é necessário que lhe sejam passados os parâmetros de *input* definidos na definição de processo associada. Na altura em que esta função é executa ocorrem os seguintes passos.

1. É criada uma instância de processo.
2. É inserida na tabela específica desta definição uma linha para esta instância de processo, onde serão também introduzidos os valores para os parâmetros de entrada.
3. É iniciada a execução da instância de processo.

A criação e arranque de uma instância tem sempre de seguir estes passos pois para introduzir os dados na tabela é necessário o Identificador produzido pela criação da instância, e para arrancar a instância é necessário que os dados já se encontrem disponíveis na base de dados.

Foi necessário também algum processamento extra para tornar estas variáveis e parâmetros disponíveis durante a execução da lógica associada ao processo.

Para os nós da definição de processo que possuam lógica associada é gerado um método na classe da respectiva definição, onde é escrita a lógica respectiva. Para o uso das variáveis e parâmetros nestes métodos tornou-se necessário gerar código que trate da leitura das variáveis da BD antes de qualquer outra lógica ser executada e que efectue a escrita no final da execução do método, de modo a efectivar as alterações às variáveis. Esta solução permite-nos ter contexto nos processos sem acrescentar complexidade ao Motor de Execução.

Isto não só permite dotar os processos de contexto, como ao fazê-lo recorrendo à base de dados permite que o seu contexto seja consultado por outras aplicação que assim o necessitem, por exemplo para apresentação de informação sobre os processos.

Embora não tenha sido efectuado nesta fase, existe ainda um ponto referente ao contexto dos processos que tem ainda de ser endereçado. Uma vez que os processos podem ter fluxos de execução paralelos é necessário abordar o acesso concorrencial a variáveis. A solução anterior não leva este problema em conta.

6. Resultados

Após detalhadas as particularidades da implementação resta apenas mostrar os resultados, ou seja, mostrar a execução de um processo de negócio.

Este capítulo inicia-se com a descrição do processo exemplo que foi utilizado para a recolha de resultados, em seguida mostram-se os resultados para o caso particular do gerador de código e posteriormente para o motor de execução.

Dada a inexistência de um componente de monitorização da execução de processos de negócio desenvolveu-se recorrendo à plataforma OutSystems uma aplicação que permite inspeccionar as tabelas da base de dados (BD) referentes ao modelo de dados do motor de execução. Esta aplicação descrita em maior detalhe no anexo 2, efectua *queries* à BD e cruza a informação resultante de maneira a apresentar uma informação formatada que torna mais simples a compreensão do conteúdo das tabelas. Passa assim a ser possível acompanhar a execução e geração dos processos, pois pode-se apresentar a informação na BD das definições de processo, incluindo nós e ligações, bem como a informação das instâncias e *Tokens* de processo. As imagens desta aplicação aparecem dentro de caixas azuis.

Seguem-se então os resultados.

6.1 Caso de Estudo

Para demonstrar a execução de processos na plataforma, escolheu-se um processo simples, mas com sentido de negócio. Assim não foram usadas todas as primitivas da linguagem, pois iria tornar o exemplo num processo irreal ou demasiado longo o que dificultaria a monitorização e apresentação dos resultados.

Foi também desenvolvida uma aplicação com um conjunto de ecrãs nos quais se simulou a interacção dos utilizadores com o processo. Estes ecrãs permitiram dar início à execução de processos e a efectuar aprovações que num caso real seriam feitas por colaboradores da empresa.

O processo e a aplicação, descritos em seguida, são propositadamente simples para facilitar a sua compreensão.

6.1.1 Processo de Negócio

O processo escolhido retrata o processamento de pedidos material de escritório de uma organização. Este é um processo simples no qual são utilizadas algumas das primitivas da linguagem

e ainda abre a possibilidade de efectuar aprovações que exemplificam o modo de interacção com os processos.

A Figura 29 mostra o processo modelado.

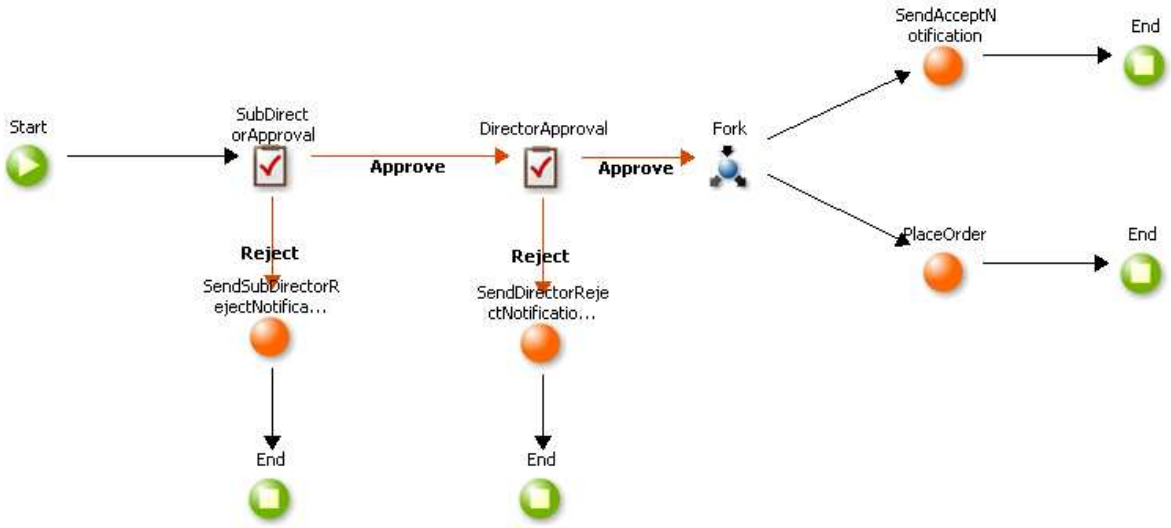


Figura 29 : Processo Material Request

Este processo começa com um nó **Start**, tal como todos os processos na linguagem de modelação adoptada pela OutSystems.

Em seguida surge a tarefa **SubDirectorApproval** que marca a aprovação do pedido por parte do subdirector. Após a avaliação da necessidade do pedido o subdirector aprova ou rejeita o pedido. Caso o pedido seja rejeitado é enviado o evento **Reject** para o processo, a tarefa **SendSubDirectorRejectNotification** notifica quem submeteu o pedido da rejeição e o processo termina, caso contrário é enviado o evento **Approve** e o processo continua.

A tarefa **DirectorAproval** pretende modelar a aprovação do pedido por parte do director, tendo em conta o orçamento disponível. Caso este rejeite, é enviado o evento **Reject** para o processo, a tarefa **SendDirectorRejectNotification** notifica quem submeteu o pedido da rejeição e o processo termina, caso contrário é enviado o evento **Approve** e o processo continua.

O nó seguinte é um **Fork** que transforma o fluxo de execução actual em dois fluxos de execução paralelos. Neste caso particular o mais natural seria as duas tarefas seguintes serem executadas em série, mas como esta alteração não afecta a essência deste processo aproveitou-se para mostrar a execução da primitiva **Fork**.

A tarefa **SendAcceptNotification** notifica quem submeteu o pedido que este foi aprovado e vai ser realizado. Após esta tarefa o fluxo de execução deste ramo termina.

A tarefa **PlaceOrder** efectiva o pedido, esta comunica com a entidade responsável por atender este tipo de pedidos, enviando um e-mail ou recorrendo a um *WebService*. Após esta tarefa o fluxo de execução deste ramo também termina.

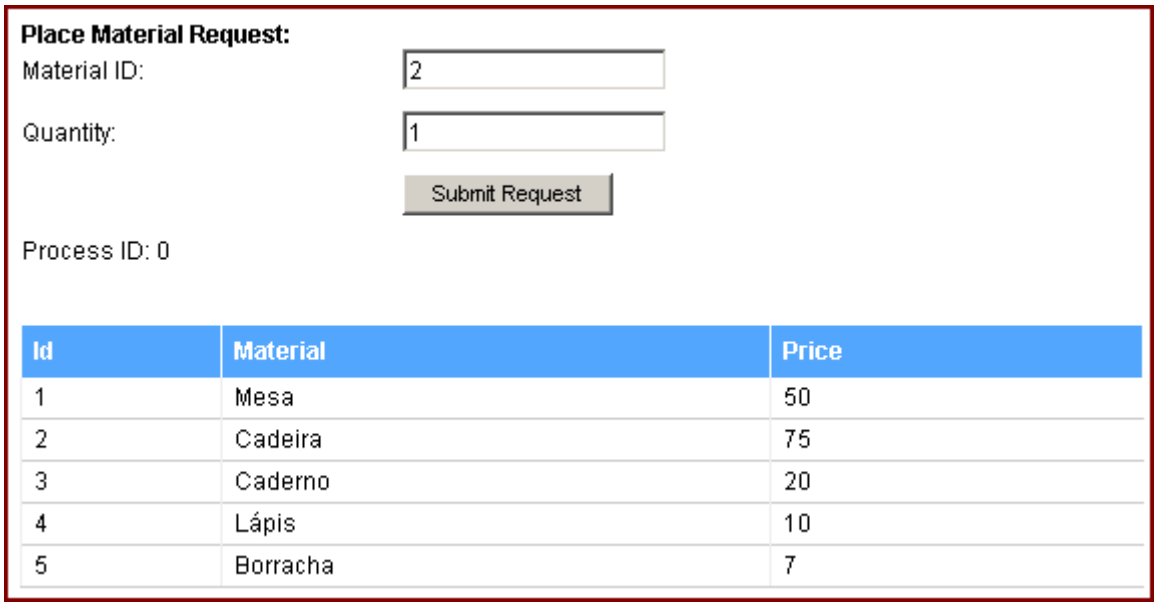
Para este exemplo não foi implementada a lógica das quatro tarefas representadas pelos nós de *AutomáticTask* pois para os três exemplos de notificações seria necessário um servidor de e-mail disponível e para a quarta um *WebService.* É no entanto de salientar que isto em nada afecta a expressividade do exemplo em causa, pois o processamento da lógica destas tarefas é efectuado da mesma forma que todas as acções já existentes na plataforma OutSystems.

6.1.2 Aplicação de Interacção

De forma a iniciarem-se processos e a realizar as aprovações que surgem no processo, criou-se uma aplicação que guarda o registo dos pedidos de material submetidos. Cada um destes pedidos corresponde a uma instância de processo. Esta aplicação recorre a 3 ecrãs que permitem a um utilizador realizar as actividades descritas anteriormente. Para evitar confusões com a aplicação de consulta de processos, as imagens desta aplicação aparecem em caixas vermelhas.

O primeiro ecrã permite a submissão de pedidos de material. A lógica associada à submissão deste tipo de pedido inclui a criação de uma instância de processo recorrendo para tal à função *CreateMaterialRequest* da *Process API*.

Para facilitar a utilização criaram-se cinco pedidos de material tipo sendo necessário apenas a introdução do identificador do tipo de pedido para efectuar a sua submissão.



Id	Material	Price
1	Mesa	50
2	Cadeira	75
3	Caderno	20
4	Lápis	10
5	Borracha	7

Figura 30 : Ecrã de submissão de pedidos

O segundo ecrã reflecte a aprovação por parte do Subdirector. Este lista todos os pedidos cujo processo está à espera de aprovação por parte do Subdirector, dando a escolha de aprovar ou rejeitar cada um deles. A lógica associada à aprovação ou reprovação de um pedido recorre à função *RaiseEventAtProcessMaterialRequest* da *Process API* que recebe o evento a ser enviado, neste caso "Approve" ou "Reject".

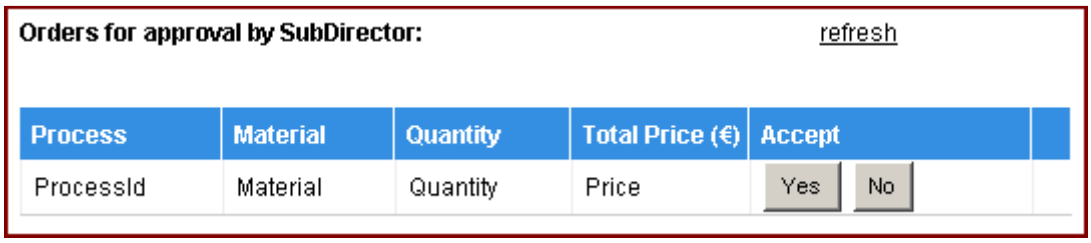


Figura 31 : Ecrã de aprovação por parte do subdirector

O terceiro ecrã reflecte a aprovação por parte do Director. Tal como o anterior este lista todos os pedidos cujo processo espera uma aprovação ou rejeição do Director. Mais uma vez a lógica associada à aprovação de um pedido recorre à função *RaiseEventAtProcessMaterialRequest* da *Process API* que recebe o evento a ser enviado.

Figura 32 : Ecrã de aprovação por parte do Director

O conjunto dos ecrãs anteriores permite a criação de várias instâncias de processo e a possibilidade de terminar estas instâncias de maneiras diferentes consoante os eventos enviados ao processo.

Esta aplicação em conjunto com a aplicação apresentada no anexo 2, permitiram a execução do processo e recolha dos resultados, que de outra forma teriam de ser recolhidos directamente da base de dados onde figuram várias definições e instâncias de processo, tornando a compreensão dos resultados bastante difícil.

6.2 Geração de Código

Após a modelação de qualquer processo no *Service Studio* (componente de modelação), este é enviado para o *HubServer* (componente de execução) através do 1-CP *(1-Click Publish)* para ser compilado e instalado.

Uma vez no *HubServer* a primeira tarefa corresponde à geração de código, onde o ficheiro com a definição de processo é processado produzindo os dois *outputs* do gerador. Inicialmente é preenchida a base de dados com os dados da definição e posteriormente gerada a classe com a lógica da definição que inclui também os Identificadores recolhidos da base de dados (BD).

6.2.1 Preenchimento da Base de Dados

Recorreu-se à aplicação de consulta da base de dados para recolher a informação inserida na BD como resultado do processamento da definição por parte do gerador de código.

Ao efectuar-se uma procura pelo nome da definição de processo *"MaterialRequest"*, obtém-se a seguinte definição:

Detalhes da Process Definition:

Id	1
Name	MaterialRequest
eSpace Version	2
Start Node	**Id:** 5 **Type:** ndStartProcess

Figura 33 : Detalhes da definição do processo *MaterialRequest*

Esta definição é composta por 12 nós. Figura 34.

Tabela de nós:

Id	Name	Process Definition	Node Type
		. . .	
8		**Id:** 1 **Name:** MaterialRequest	ndEndProcess
9		**Id:** 1 **Name:** MaterialRequest	ndEndProcess
10	PlaceOrder	**Id:** 1 **Name:** MaterialRequest	ndAutoTask
11	SubDirectorApproval	**Id:** 1 **Name:** MaterialRequest	ndTask
12	DirectorApproval	**Id:** 1 **Name:** MaterialRequest	ndTask

Figura 34 : Exemplo de nós da definição do processo *MaterialRequest*

Na Figura 35 mostram-se ainda algumas das ligações que ligam os nós anteriores.

Tabela de Transitions:

Id	Name	From Node	To Node	Process Definition	Event
			. . .		
7		**Id:** 10 **Type:** ndAutoTask **Name:** PlaceOrder	**Id:** 9 **Type:** ndEndProcess	**Id:** 1 **Name:** MaterialRequest	False
8	Reject	**Id:** 11 **Type:** ndTask **Name:** SubDirectorApproval	**Id:** 6 **Type:** ndAutoTask **Name:** SendSubDirectorRejectNotification	**Id:** 1 **Name:** MaterialRequest	True
9	Approve	**Id:** 11 **Type:** ndTask **Name:** SubDirectorApproval	**Id:** 12 **Type:** ndTask **Name:** DirectorApproval	**Id:** 1 **Name:** MaterialRequest	True
10	Approve	**Id:** 12 **Type:** ndTask **Name:** DirectorApproval	**Id:** 1 **Type:** ndFork	**Id:** 1 **Name:** MaterialRequest	True
11	Reject	**Id:** 12 **Type:** ndTask **Name:** DirectorApproval	**Id:** 2 **Type:** ndAutoTask **Name:** SendDirectorRejectNotification	**Id:** 1 **Name:** MaterialRequest	True

Figura 35 : Exemplo de ligações entre nós da definição do processo *MaterialRequest*

Juntamente com as tabelas anteriores é também devolvida a tabela de tipos que já se encontrava previamente preenchida.

Tabela de Tipos:

Id	Name
ndAutoTask	AutoTask
ndEndProcess	EndProcess
ndFork	Fork
ndJoin	Join
ndProcessIf	ProcessIf
ndProcessSwitch	ProcessSwitch
ndStartProcess	StartProcess
ndSubProcess	SubProcess
ndTask	Task

Figura 36 : Tabela de tipos preenchida

Com um olhar cuidado sobre a tabela de nós, consegue-se encontrar, por exemplo, o nó correspondente à tarefa de aprovação do director com o identificador 12 e pela consulta da tabela de ligações, vemos que esta tem duas ligações de saída, em que a primeira, com o identificador 10, liga a um nó do tipo *Fork* e possui um evento *"Approve"* e a segunda com o identificador 11 liga a um nó com o nome *"SendDirectorRejectNotification"* e possui um evento *"Reject"*.

O mesmo raciocínio pode ser usado para verificar os restantes nós e ligações.

6.2.2 Geração da Classe

O segundo resultado da geração de código é um ficheiro onde irá figurar a classe gerada que contém a definição e a lógica associada ao processo. Esta classe vai ser usada pelo motor de execução.

Como se pode ver pela Figura 37 todas as classes geradas estendem a classe *ProcessDefinition*, onde foram definidos os métodos comuns a todas as definições. No construtor da classe é atribuído o identificador criado pela base de dados e gerado o código que em execução criará os nós e as ligações desta definição.

```
public class Process_MaterialRequestDefinition: ProcessDefinition {

    private static Process_MaterialRequestDefinition _definition =
                                        new Process_MaterialRequestDefinition();

      private Process_MaterialRequestDefinition() {
            ID = 1;

            Nodes = new Hashtable();
            ...
            Node node2533 = new Start(5);
            Nodes.Add(node2533.ID, node2533);
            FirstNode = node2533;
            Node node3247 = new Activity(6,
                  new ActivityPreparation(Activity_node3247_Preparation));
            Nodes.Add(node3247.ID, node3247);
            ...

            new Transition(node3253, node2578);
            new Transition(node2553, node3247, new Event("Reject"));
            ...
      }
      ...
```

Figura 37 : Construtor da classe da definição gerada

Os nós também recebem o identificador que lhes foi atribuído pela base de dados e caso possuam lógica associada recebem um *delegate* que permitirá invocar o método onde essa lógica foi gerada.

As ligações recebem respectivamente os nós de origem e de destino que as compõem. Na criação das ligações em tempo de execução, as ligações são adicionadas à lista de ligações do nó origem.

Para os nós que possuam lógica associada, devido à sua semântica ou porque foi definida pelo utilizador, como é o caso das *AutomaticTask's*, é gerado um método que inclui o código resultante.

A Figura 38 apresenta um exemplo para uma das *AutomaticTask's* do processo utilizado.

```
public static string Activity_node3247_Preparation(int inParamProcessId,
                                                   int inParamTokenId) {
      ...
      return null;
}
...
```

Figura 38 : Métodos associados com a lógica das tarefas

No final são definidos três métodos que permitem respectivamente o acesso à definição de processo, a criação de instâncias de processo para esta definição e a criação de instâncias desta definição como sub-processo. Estes métodos são apresentados na Figura 39.

```
      public static void RegisterDefinition() {
            ProcessEngine.ProcessDefinitions.Add(_definition.ID,
                                                 _definition);
      }

      public static int CreateInstance() {
            return ProcessRuntime.CreateProcess(_definition);
      }

      public static int CreateSubInstance(int inParamTokenId, int Node) {
            return ProcessRuntime.CreateProcess(_definition,
                                                inParamTokenId, Node);
      }
}
```

Figura 39 : Métodos para criação de instâncias de processo

O anexo 3 apresenta o output completo da geração para o processo anterior.

6.3 Execução

Após a fase de geração e já com as classes e dados que implementam as definições de processos, segue-se a criação e execução de instâncias de processo.

A execução do processo dá-se em 5 fases:

1. Criação da instância.
2. Aprovação do Subdirector.
3. Aprovação do Director.
4. Criação dos fluxos paralelos.
5. Terminação da execução.

Na **fase 1** deste exemplo usa-se o ecrã de submissão de pedidos, descrito anteriormente, que irá submeter um pedido e criar uma instância do processo que vai acompanhar a duração do pedido. Assim na Figura 40 apresenta-se a criação de um pedido do tipo 3 para três cadernos.

É apresentado o identificador da instância de processo criada para que se possa seguir a sua evolução.

Após a criação, o processo executa-se até encontrar um ponto de paragem, neste caso particular a paragem ocorre devido à necessidade de aprovação por parte do subdirector.

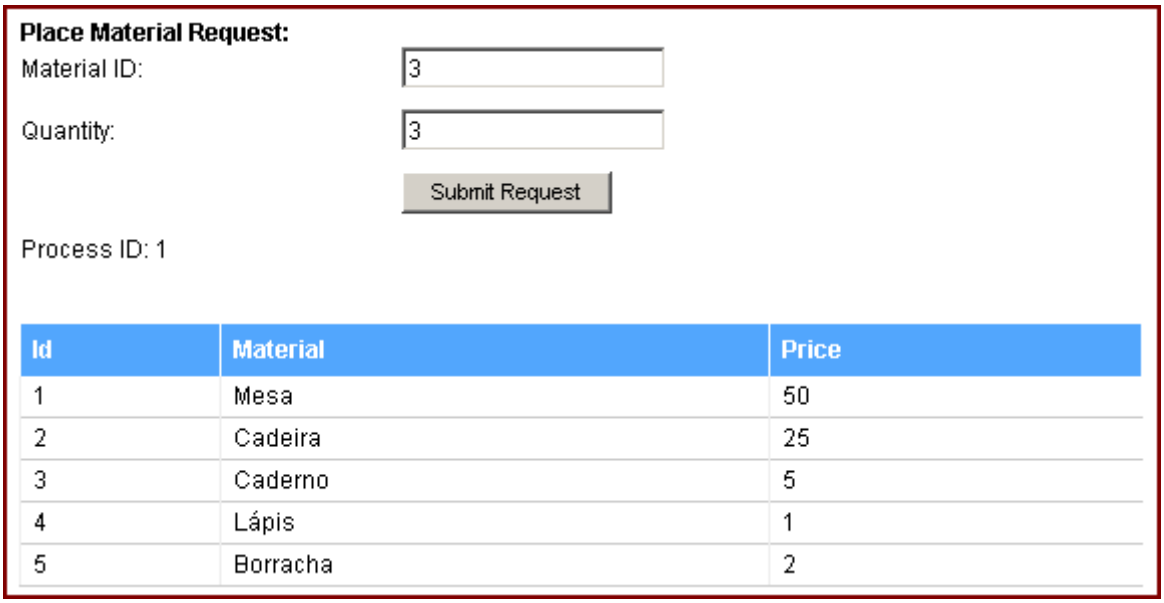
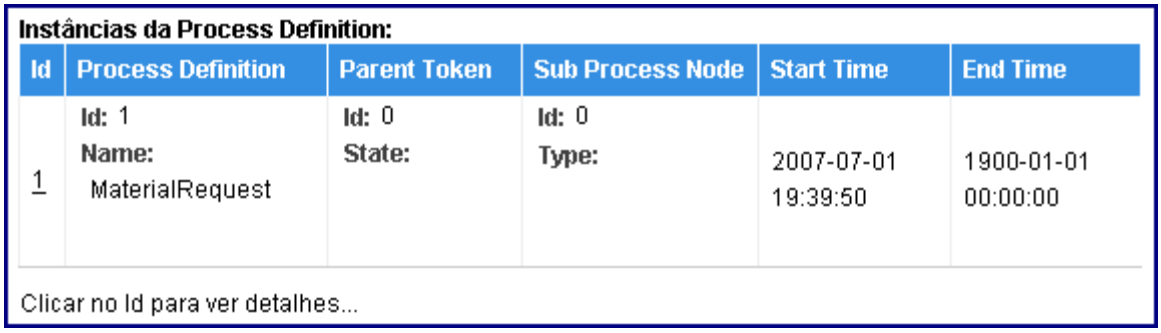


Figura 40 : Criação de um pedido de material

Recorrendo à aplicação de consulta podemos ver uma listagem das instâncias existentes que executam a definição de processo *"MaterialRequest".* Na Figura 41 pode-se ver que a instância criada anteriormente figura na lista de instâncias que executam esta definição.



Instâncias da Process Definition:

Id	Process Definition	Parent Token	Sub Process Node	Start Time	End Time
1	**Id:** 1 **Name:** MaterialRequest	**Id:** 0 **State:**	**Id:** 0 **Type:**	2007-07-01 19:39:50	1900-01-01 00:00:00

Clicar no Id para ver detalhes...

Figura 41 : Instância de processo criada

É também possível ver a listagem dos *Tokens* para a instância criada. Uma vez que nesta fase da execução o processo só tem um fluxo de execução, só existe um *Token*. É também possível ver que o *Token* se encontra no nó "*SubDirectorApproval",* onde é esperada a aprovação do director.

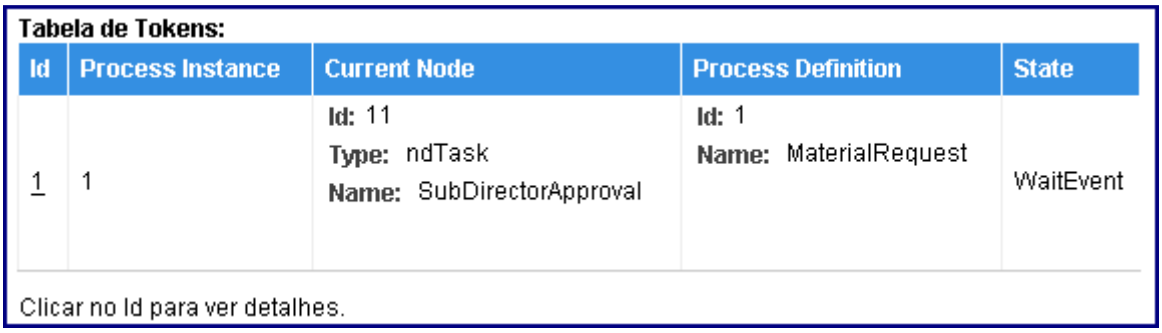


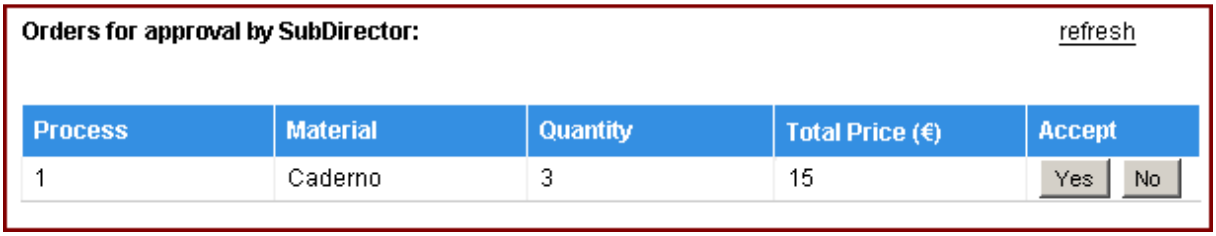
Tabela de Tokens:

Id	Process Instance	Current Node	Process Definition	State
1	1	**Id:** 11 **Type:** ndTask **Name:** SubDirectorApproval	**Id:** 1 **Name:** MaterialRequest	WaitEvent

Clicar no Id para ver detalhes.

Figura 42 : *Token* à espera do evento de aprovação do subdirector

Na **fase 2** usa-se o ecrã de aprovação do subdirector. Neste ecrã são listados os pedidos que estão para aprovação por parte do subdirector. Para cada pedido é dada a hipótese de aprovação ou rejeição. Na Figura 43 é apresentado o pedido criado anteriormente.



Orders for approval by SubDirector: refresh

Process	Material	Quantity	Total Price (€)	Accept
1	Caderno	3	15	Yes No

Figura 43 : Pedido à espera de aprovação do subdirector

Selecciona-se a opção de aprovação que irá enviar o evento *"Approve"* para a instância de processo, que retoma a sua execução até ao próximo ponto de paragem, a aprovação do Director.

Recorrendo à aplicação de consulta podemos ver que o *Token* prosseguiu a sua execução. O nó corrente do *Token* passou a ser o nó "*DirectorApproval",* onde desta vez é esperada a aprovação do director.

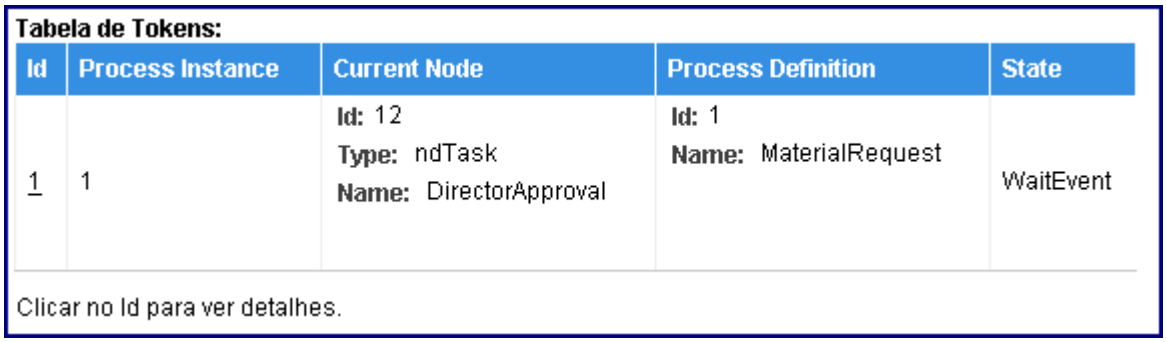


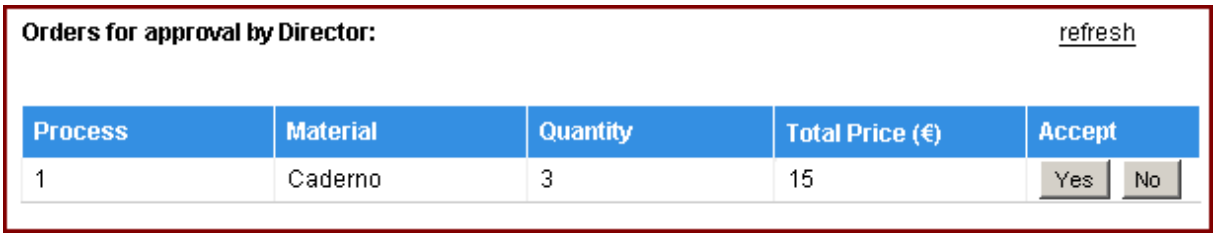
Tabela de Tokens:

Id	Process Instance	Current Node	Process Definition	State
1	1	**Id:** 12 **Type:** ndTask **Name:** DirectorApproval	**Id:** 1 **Name:** MaterialRequest	WaitEvent

Clicar no Id para ver detalhes.

Figura 44 : *Token* à espera do evento de aprovação do director

Na **fase 3** surge o ecrã de aprovação do director. Tal como no caso anterior, este ecrã lista os pedidos que estão para aprovação desta vez por parte do director. A Figura 45 apresenta o pedido que espera a aprovação do director.



Orders for approval by Director: refresh

Process	Material	Quantity	Total Price (€)	Accept
1	Caderno	3	15	Yes No

Figura 45 : Pedido à espera de aprovação do director

Selecciona-se a opção de aprovação que irá enviar o evento *"Approve"* para a instância de processo. Esta retoma a sua execução até ao próximo ponto de paragem que neste caso será o final do processo.

Na **fase 4** é executado o nó *Fork* que faz com que o fluxo de execução actual termine e dê origem a dois novos fluxos de execução que irão executar paralelamente as tarefas restantes chegando à **fase 5** da terminação do processo.

Recorrendo mais uma vez à aplicação de consulta podemos ver a listagem de *Tokens* para a instância de processo na Figura 46. O *Token* com identificador 1 apresentado nas fases anteriores encontra-se no estado *"Finish"* de terminado e surgiram dois novos *Tokens* responsáveis pela execução das duas tarefas paralelas. Estes dois novos *Tokens* também já executaram as respectivas tarefas e encontram-se no estado *"Finish".*

Tabela de Tokens:

Id	Process Instance	Current Node	Process Definition	State
1	1	**Id:** 1 **Type:** ndFork	**Id:** 1 **Name:** MaterialRequest	Finished
2	1	**Id:** 3 **Type:** ndEndProcess	**Id:** 1 **Name:** MaterialRequest	Finished
3	1	**Id:** 8 **Type:** ndEndProcess	**Id:** 1 **Name:** MaterialRequest	Finished

Clicar no Id para ver detalhes.

Figura 46 : Lista de *Tokens* após o final da execução do processo

Com a terminação da execução de todos os *Tokens*, a instância de processo é marcada como terminada sendo-lhe atribuído um tempo de terminação.

Instâncias da Process Definition:

Id	Process Definition	Parent Token	Sub Process Node	Start Time	End Time
1	**Id:** 1 **Name:** MaterialRequest	**Id:** 0 **State:**	**Id:** 0 **Type:**	2007-07-01 19:39:50	2007-07-01 19:59:50

Clicar no Id para ver detalhes...

Figura 47 : Instância do processo após o final da execução do processo

Após a execução todos os registos se mantêm na base de dados para efeitos de consulta.

A execução dos nós que não foram utilizados no exemplo é semelhante aos apresentados, a sua execução afecta os *Tokens* e a instância do processo, que devido à sua semântica podem afectar o resultado final da execução do processo.

7. Conclusão

Os processos de negócio surgiram nas organizações de hoje em dia como a forma de estruturar o negócio de forma a atingir a flexibilidade que permitirá responder às incessantes alterações das condições de mercado.

Os sistemas das organizações encontram-se cada vez mais interligados e é através da orquestração de actividades em processos de negócio que estes são geridos e adaptados para a produção de valor para a organização.

As ferramentas conhecidas como *Business Process Management Suites* ou *BPMS's* oferecem um conjunto de funcionalidades que permitem abordar toda a gestão dos processos de negócio numa organização, desde a criação, integração com outros sistemas, monitorização de processos, entre outros. Embora estas ferramentas ofereçam uma flexibilidade antes inexistente nas organizações requerem ainda um elevado grau de programação baixo nível, o que faz com que estejam ainda longe do ideal em que após a modelação de um processo pouco ou nenhum trabalho acrescido é necessário para que se possa proceder à execução deste.

A OutSystems com a sua inovadora metodologia e plataforma, permite desenvolver aplicações visualmente recorrendo a um componente de modelação. Isto torna as suas aplicações extremamente flexíveis, pois após a modelação de toda a aplicação esta encontra-se pronta para ser executada. Apesar da flexibilidade oferecida ser bastante proveitosa para as organizações o facto da plataforma OutSystems não abordar processos de negócio, leva que a sua plataforma não seja escolha para muitas organizações, pois tendo em conta a estruturação das organizações em torno dos processos de negócio pode levar a que as aplicações desenvolvidas fiquem desalinhadas do pretendido.

Neste sentido surgiu a proposta de incluir processos de negócio na plataforma OutSystems, mais concretamente, componentes que permitissem modelar e executar e posteriormente monitorizar processos de negócio. Só com a combinação destes elementos se consegue alinhar esta plataforma com as novas necessidades que surgiram no mercado.

A inclusão de processos de negócio na plataforma permite que esta seja capaz de desenvolver processos de negócio e aplicações que interajam com os processos recorrendo ao desenvolvimento visual, sem a necessidade de programação escrita. Isto torna o desenvolvimento bastante mais rápido tal como é necessário pelas organizações, torna mais fácil a alteração oferecendo a flexibilidade necessária e ainda facilitando a compreensão das aplicações desenvolvidas pelos elementos do negócio. Para além das vantagens anteriores o desenvolvimento de processos de negócio segue a mesma aproximação que o desenvolvimento de aplicações, ou seja, após a modelação é necessário apenas 1 clique (1-Click-Publish) até este estar pronto para execução. Embora não se possa negligenciar o desenvolvimento de toda da lógica associada aos processos,

esta aproximação torna a criação e execução de processos bastante mais ágil e fácil que as aproximações de outras ferramentas existentes.

7.1 Trabalho Realizado

No decorrer deste trabalho abordaram-se possíveis soluções para suportar a execução de processos na plataforma e após escolhida a melhor solução procedeu-se à sua implementação.

De maneira a que os processos fossem um componente integral da plataforma e não apenas um complemento, optou-se por seguir a solução o mais próximo possível da estrutura da plataforma já existente. Neste sentido o gerador de código e o ambiente runtime da componente de execução da plataforma foram os componentes que necessitaram de maior atenção para permitir o suporte de processos.

O gerador de código é um componente inexistente nas demais ferramentas do mercado e que permite que a plataforma OutSystems ofereça uma maior flexibilidade. Este componente permite que através do resultado do desenvolvimento visual das aplicações, no qual agora foi incluído a modelação de processos de negócio, seja produzido automaticamente todo o material necessário à execução, que na maioria das ferramentas teria de ser produzido manualmente pelo utilizador. Este teve de ser estendido para abordar o processamento dos processos modelados produzindo o material necessário.

No ambiente runtime teve de se incluir um motor de execução de processos, tal como acontece com as restantes ferramentas. O facto de este motor fazer parte integral da plataforma traz vantagens pois não só permite uma execução mais eficaz visto que não necessita de integrar com outros componentes, mas também permite que em tempo de execução haja uma forte ligação entre os processos e os restantes elementos das aplicações na plataforma. Seguindo uma visão macroscópica pode-se afirmar que este motor de execução é semelhante a uma máquina de estados que gere a evolução dos processos no decorrer da sua execução.

Obteve-se assim a capacidade de execução de processos de negócio que pelo facto de estarem integrados na plataforma, partilham das vantagens e facilidades que esta oferece. O facto de toda a execução se basear largamente no modelo de dados abre a possibilidade para o uso da informação gerada pela execução de processos por sistemas externos.

É certo que a solução desenvolvida não aborda todos os aspectos existentes em torno dos processos de negócio. O desenvolvimento de raiz de uma solução capaz de abordar todos os aspectos relevantes à execução de processos de negócios é um problema demasiado amplo para ser possível abordá-lo de uma só vez, neste sentido optou-se por abordar exclusivamente a execução de processos no sentido da evolução do estado ao longo da execução, por ser a função inexistente com a qual já seria possivel desenvolver soluções com processos. Para depois foram deixados conceitos como a gestão de utilizadores e inboxes que apesar de não serem abordados directamente no seio

do desenvolvimento de processos podem ser abordados por outros elementos já existentes na plataforma, que utilizados em paralelo com os processos permitem o desenvolvimento de soluções completas.

7.2 Trabalho Futuro

Após o trabalho apresentado, a plataforma OutSystems suporta o desenvolvimento de processos de negócio ou de aplicações que façam uso de processos de negócio. Contudo até esta plataforma possibilitar uma gestão e execução completa de processos de negócio existem ainda certos aspectos que necessitam de ser endereçados.

Um dos aspectos a desenvolver de futuro prende-se com os utilizadores e respectivas inboxes. Como dito anteriormente existem elementos na plataforma que permitem abordar estes elementos, no entanto pretende-se a inclusão destes no seio dos processos de negócio pois estes conceitos estão fortemente ligados aos processos. Uma abordagem possível é criação de um modo que permita a partilha da gestão destes conceitos no seio dos processos mas também nos restantes pontos da plataforma.

Outro aspecto prende-se com a informação sobre a execução de processos que é guardada na base de dados (BD). O estado actual permite consultar o estado actual de qualquer instância ou *Tokens* de processo, contudo não é possível consultar a evolução da execução, ou seja, não é guardada na BD qual o conjunto de tarefas (caminho) que foram executadas por cada instância. Esta informação é bastante importante nomeadamente para a monitorização e gestão dos processos, pelo que é se deve criar um sistema de *log* que guarde toda esta informação. Uma possível solução é a criação de uma nova tabela na BD onde se registem todas as operações efectuadas sobre as instâncias ou *Tokens* em execução.

Um problema comum que cedo se encontrou prende-se com a gestão de versões de processos em execução, mais concretamente o que fazer a uma instância em execução cuja definição de processo que foi alterada. No estado actual, alterações a uma definição dão origem a uma versão diferente que não afecta as instâncias de processo das versões anteriores. Esta solução pode não ser razoável em certos casos, como por exemplo, uma nova versão cujas alterações realizadas às aplicações do processo retiraram interfaces necessárias para o utilizador fazer avançar o processo da versão anterior. É necessário efectuar mais estudos para compreender qual a solução esperada nestes casos.

O aspecto final a abordar prende-se com a completude da plataforma. De maneira a ser possível retirar conclusões sobre a execução de processos é necessário um componente de monitorização de processos. Este componente permitirá que se faça uma real gestão dos processos executados sobre esta plataforma permitindo uma avaliação dos resultados que levem à toma de medidas.

8. Referências

[1] Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New Directions on Agile Methods: A comparative analisys. *Proceedings of the 25th International Conference on Software Engineering* , 244--254.

[2] Active Endpoint. (2007, June). *Active Endpoint Process Engine Architecture.* Retrieved from http://www.active-endpoints.com/open-source-architecture.htm

[3] Azoff, M. (2007). OutSystems Platform 4.0. Butler Group.

[4] Brown, P., & Szefler, M. (2003). BPEL for Programmers and Architects. *FiveSight Technologies Inc.* FiveSight Technologies Inc.

[5] Cardoso, J., Bostrom, R., & Sheth, A. (2004). Workflow Management Systems and ERP Systems: Differences, Commonalities, Applications. *Information Technology and Management , 5* (3), 319--338.

[6] Ferreira, Diogo (February 2004). Workflow Management Systems Supporting the Engineering of Business Networks. *FEUP.* PhD Thesis.

[7] Ferreira, Diogo, & Ferreira, J. J. P. (June 2004). Developing a Reusable Workflow Engine. *Journal of Systems Architectures , vol. 50, no. 6*, pp. 309-324.

[8] Graça, F. (2007). Modelação de Processos de Negócio. *Instituto Superior Técnico.* Lisbon: Msc Thesis.

[9] Graça, F., Almeida, H., & Silva, M. M. (2007). Business Oriented Process Modeling Notation. *DET2007.*

[10] Havey, M. (2005). *Essential Business Process Modeling.* O'Reilly.

[11] Hollingsworth, D., & others. (1995). *The Workflow Reference Model.* Workflow Management Coalition.

[12] Jungel, M., Kindler, E., & Weber, M. (2000). The Petri Net Markup Language. *Petri Net Newsletter , 59*, 24--29.

[13] Kloppmann, M., Koening, D., Leymann, F., Pfau, G., Rickayzen, A., von Riegen, C., et al. (2005, July). WS-BPEL Extension for People – BPEL4People. *Joint white paper, IBM and SAP* .

[14] List, B., & Korherr, B. (2006). An evaluation of conceptual business process modelling languages. *Proceedings of the 2006 ACM symposium on Applied computing* , 1532--1539.

[15] McCoy, D. (2002). Business Activity Monitoring: Calm Before the Storm. *Gartner Research Note LE-15-9727* .

[16] Michelson, B. M. (2005). *Business Process Execution Language (BPEL) Primer.* Patricia Seybold Group.

[17] Miers, D. (2006). Best practice BPM. *Queue , 4* (2), 40--48.

[18] Morkel, W. H., Mourie, D. G., & Watson, B. W. (2003). Standards and Agile Software Development. *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology* , 178--188.

[19] OASIS. (2007, January). *BPEL*. Retrieved from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[20] OASIS. (2007, June). *BPEL Specification 2.0.* Retrieved from http://www.oasis-open.org/committees/download.php/18714/wsbpel-specification-draft-May17.htm

[21] *OutSystems*. (2007, June). Retrieved from http://www.outsystems.com

[22] OutSystems. (2007). OutSystems Agile Methodology.

[23] OutSystems. (2007). OutSystems Platform Technical Overview.

[24] OutSystems. (2006). The Agile Software Platform Company.

[25] Ouyang, C., van der Aalst, W. M., Dumas, M., & Hofstede, A. H. (2006). Translating BPMN to BPEL. (IEEE, Ed.)

[26] Rising, L., & Janoff, N. S. (2000). The Scrum Software Development Process for Small Teams. *Software, IEEE , 17* (4), 26--32.

[27] Russel, N., van der Aalst, W. M., Hofstede, A. H., & Wohed, P. (2006). On the suitability of UML 2.0 activity diagrams for business process modelling. *Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling , 53*, 95--104.

[28] Schwaber, K. (1995). Scrum Development Process. *OOPSLA'95 Workshop on Business Object Design and Implementation* .

[29] Silver, B. (2005). *The 2006 BPMS Report: BEA AquaLogic BPM Suite v5.5.* BPMInstitute.

[30] Silver, B. (2005). *The 2006 BPMS Report: IBM WebShpere BPM Suite v6.0.* BPMInstitue.

[31] Silver, B. (2005). *The 2006 BPMS Report: Savvion BusinessManager v6.5.* BPMInstitute.

[32] Smith, H., & Fingar, P. (2004). BPM is Not About People, Culture and Change, It's About Technology.

[33] Smith, H., & Fingar, P. (2003). *Business Process Management The Third Wave.* Meghan-Kiffer Press.

[34] Smith, H., & Fingar, P. (2004). The Third Wave: BPM 2004.

[35] The Workflow Management Coalition. (2005). *XPDL Specification 2.0.* Retrieved from http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf

[36] van der Aalst, W. M. (1998). The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers , 8* (1), 21--66.

[37] van der Aalst, W. M., Dumas, M., Hostede, A. H., Russel, N., Verbeek, H. M., & Wohed, P. (2005). Life After BPEL. *WS-FM , 3670*, 35--50.

[38] Verner, L. (2004). BPM, The Promise and the Challenge. *Queue , 2* (1), 82--91.

[39] White, S. A. (2004). Introduction to BPMN. *IBM Cooperation* .

Anexos

1. BPEL

1.1 Linguagem BPEL

Tabela 3 : Elementos BPEL **[4], [16]**

Grupo	Elemento	Descrição
Declarações	**<process>**	Elemento raiz
	<partnerLink>	Descreve colaboradores, incluindo *role* e operações destino. Normalmente são *Web Services. (WSDL port type)*
	<variable>	Variáveis no processo
Actividades Básicas	**<receive>**	Bloqueia e espera por uma mensagem de um colaborador.
	<reply>	Responde ao colaborador.
	<invoke>	Invoca um *Web Service* de um colaborador. Permite invocações síncronas ou assíncronas.
	<assign>	Cópia de dados entre variáveis.
	<throw>	Gera uma falta.
	<wait>	Suspende a execução por um determinado período de tempo.
	<empty>	No-op. Operação vazia.
Actividades Estruturadas	**<sequence>**	Executa um conjunto de actividades pela ordem especificada.
	<switch>	Elemento de escolha conforme vários casos.
	<while>	Ciclo. Repete-se enquanto uma condição for verdadeira.
	<pick>	Bloqueia a execução e espera por uma mensagem ou alarme.
	<flow>	Executa um conjunto de actividades concorrentemente.
	<eventHandlers>	Recebe uma mensagem ou alarme, sem bloquear a execução.
Actividades de Gestão	**<scope>**	Determina o alcance para *variable's*, *faultHandler's*, *compensate's*, *correlation's*.
	<faultHandlers>	Design o tratamento de excepções consoante determinadas condições.
	<compensate>	Em caso de falha executa lógica de compensação, normalmente para desfazer actividades previamente realizadas
	<correlations>	Associa mensagens com instâncias de processos específicas.
	<terminate>	Termina e destrói o processo.

1.2 Exemplo de um Processo em BPEL

Figura 48 : Exemplo de um Processo em BPEL **[16]**

2. Aplicação de Visualização de Resultados

Dada a inexistência de um meio de monitorizar os processos em execução foi necessário desenvolver uma aplicação que através da consulta da base de dados permitisse efectuar uma monitorização dos processos em execução.

Esta aplicação desenvolvida em cima da plataforma OutSystems, é composta por cinco ecrãs que mostram diferentes níveis de informação. Os ecrãs apresentados são os seguintes:

1. Ecrã de procura de definições de processo.
2. Ecrã de apresentação de detalhes de uma definição de processo.
3. Ecrã de listagem de instâncias de processo para uma determinada definição.
4. Ecrã de apresentação de detalhes de uma instância de processo.
5. Ecrã de apresentação de detalhes de um *Token* de processo.

Sempre que possível a informação apresentada nestes ecrãs resulta do cruzamento, da informação presente na base de dados, de modo a que a informação apresentada seja facilmente compreensível pelo utilizador.

O conjunto destes ecrãs permite obter uma visão completa dos vários elementos presentes na execução dos processos e desta forma monitorizar a sua evolução enquanto estes se executam.

2.1 Procura de Definições de Processo

Este ecrã permite procurar definições de processo por nome. Após a procura são listadas as várias definições com o mesmo nome. Versões diferentes da mesma definição de processo são listadas como definições diferentes pois apresentam identificadores diferentes.

Este é o ponto inicial da aplicação, a partir deste ponto podem-se consultar os detalhes da definição ou as instâncias que se encontram a executar esta definição.

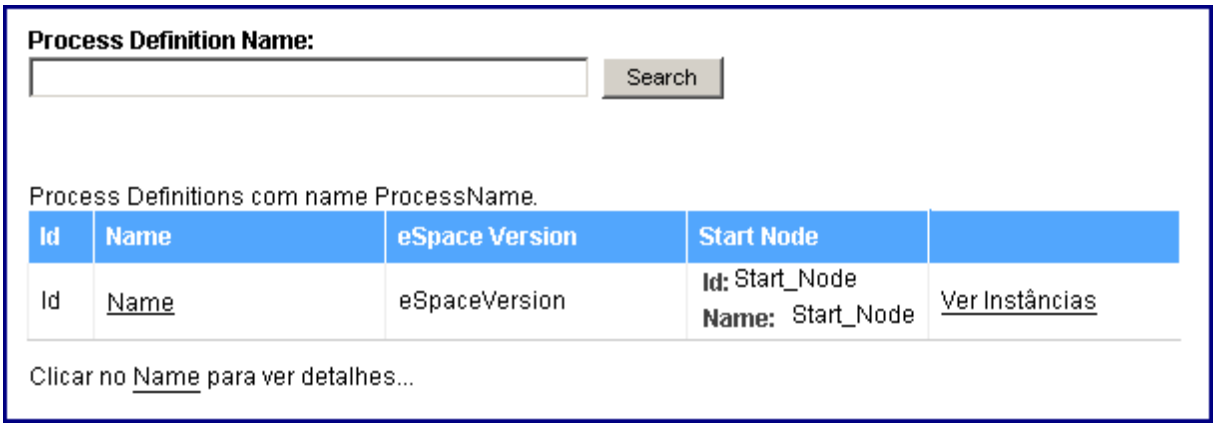


Figura 49 : Procura de definições de processo

2.2 Detalhes de Definição de Processo

Este ecrã permite consultar os detalhes das definições de processo. São apresentados os dados da definição de processo, bem como duas tabelas com os nós e as transições que compõem a definição. Para além destes dados é sempre apresentada a tabela de tipos que centraliza os vários tipos de nós da linguagem.

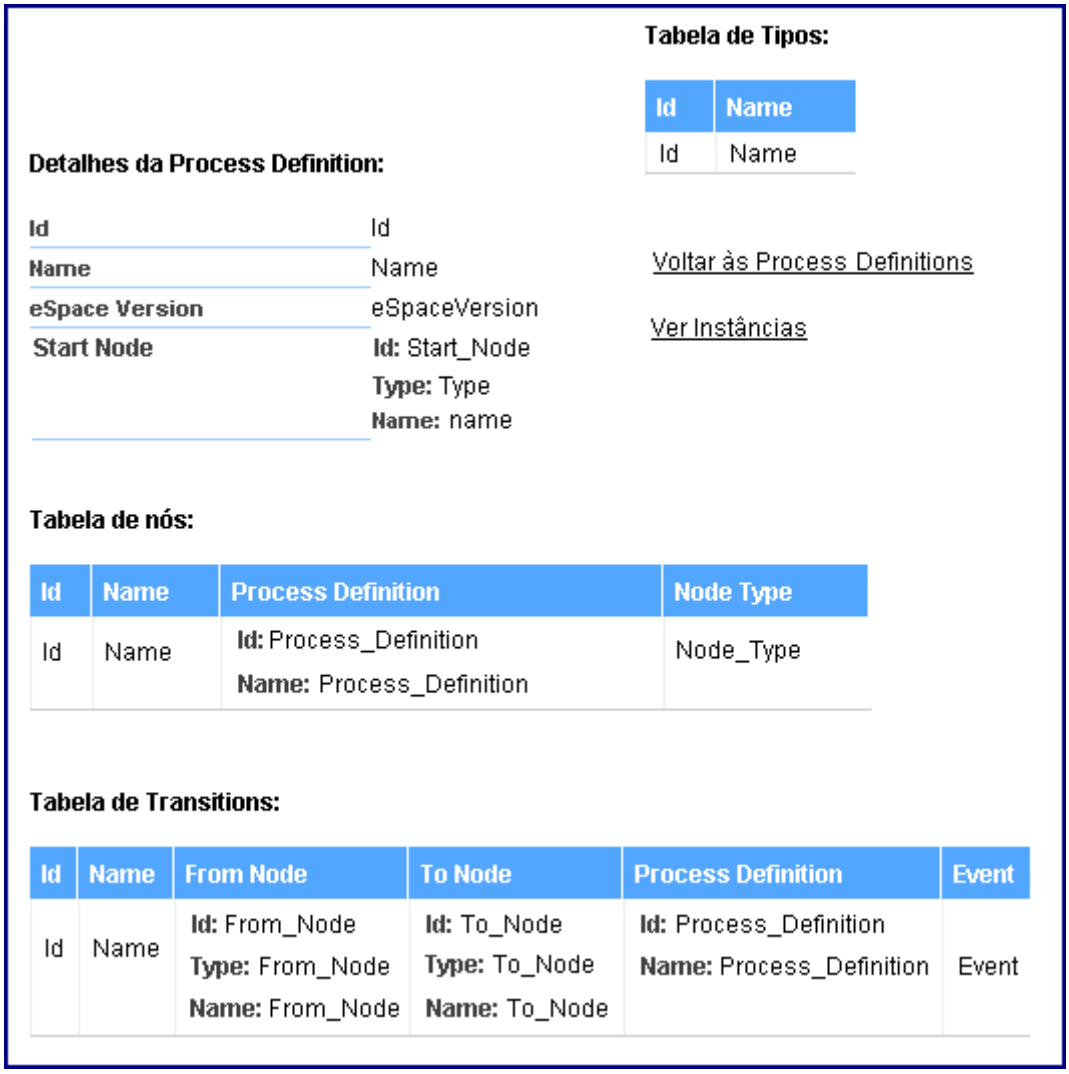


Figura 50 : Detalhes de um definição de processo

2.3 Listagem de Instâncias de Processo

Este ecrã lista as várias instâncias de processo de uma definição particular. Para além dos detalhes da definição de processo é apresentada uma tabela que lista todas as instâncias de processo que estão a executar esta definição.

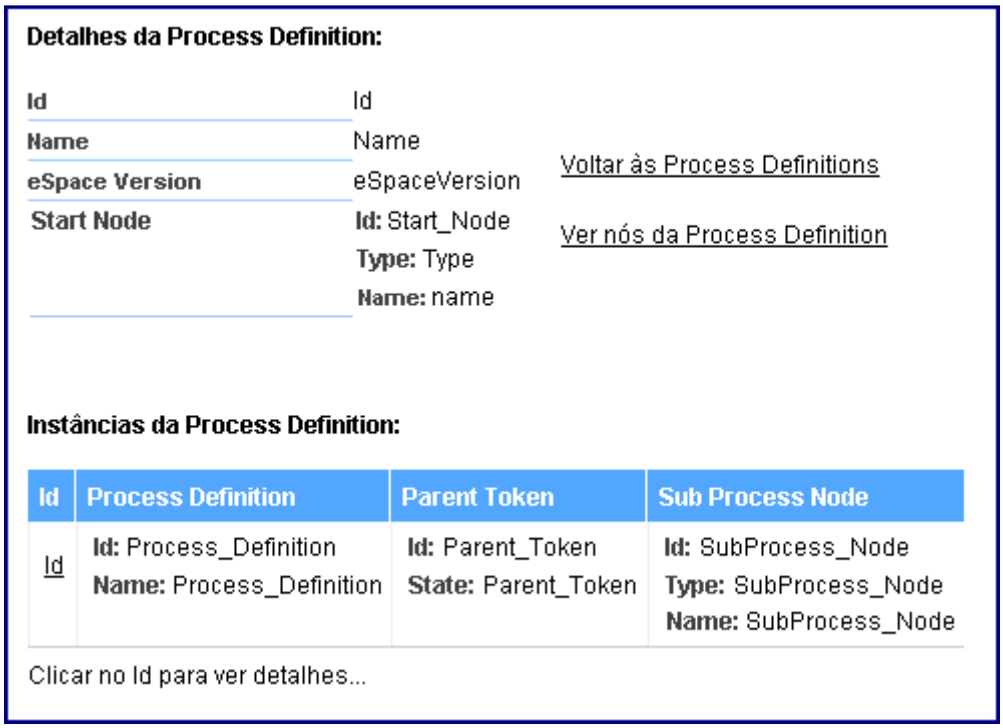


Figura 51 : Listagem das instâncias de processo de uma definição

2.4 Detalhes de Instância de Processo

Este ecrã permite consultar os detalhes das instâncias de processo. São apresentados os dados da instância de processo e ainda uma tabelas com os *Tokens* que compõem esta instância. São sempre apresentados todos os *Tokens* independente do estado em que se encontram.

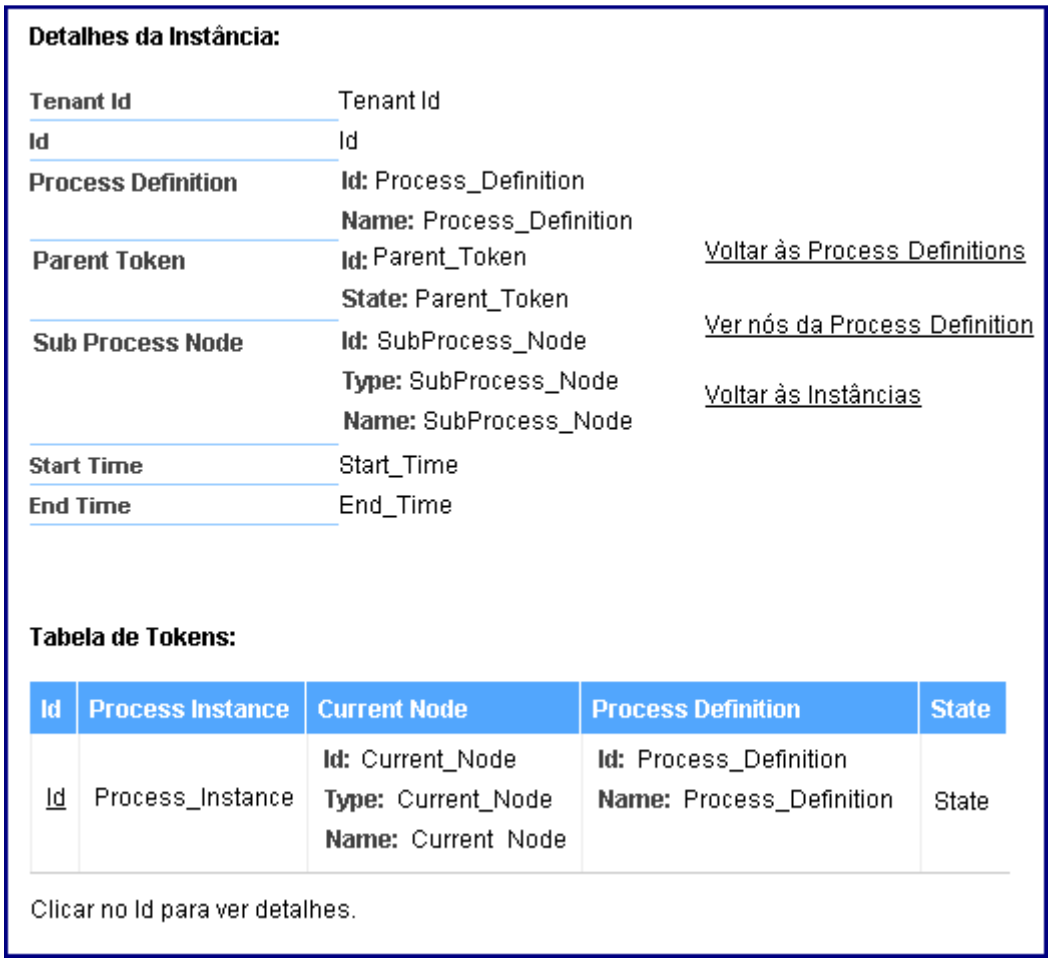


Figura 52 : Detalhes de uma instância de processo

2.5 Detalhes de *Token* de Processo

Este ecrã permite consultar os detalhes de um *Token* particular, apresentando todos os dados referentes a esse mesmo *Token*.

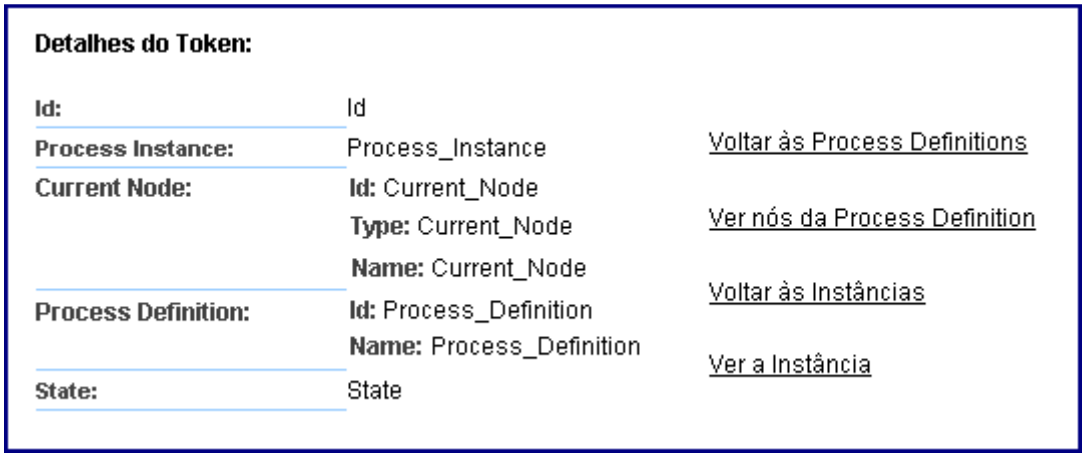


Figura 53 : Detalhes de um *Token* de processo

3. Output da Geração do Processo Exemplo

3.1 Definição na Base de Dados

Tabela de nós:

Id	Name	Process Definition	Node Type
1		**Id:** 1 **Name:** MaterialRequest	ndFork
2	SendDirectorRejectNotification	**Id:** 1 **Name:** MaterialRequest	ndAutoTask
3		**Id:** 1 **Name:** MaterialRequest	ndEndProcess
4	SendAcceptNotification	**Id:** 1 **Name:** MaterialRequest	ndAutoTask
5		**Id:** 1 **Name:** MaterialRequest	ndStartProcess
6	SendSubDirectorRejectNotification	**Id:** 1 **Name:** MaterialRequest	ndAutoTask
7		**Id:** 1 **Name:** MaterialRequest	ndEndProcess
8		**Id:** 1 **Name:** MaterialRequest	ndEndProcess
9		**Id:** 1 **Name:** MaterialRequest	ndEndProcess
10	PlaceOrder	**Id:** 1 **Name:** MaterialRequest	ndAutoTask
11	SubDirectorApproval	**Id:** 1 **Name:** MaterialRequest	ndTask
12	DirectorApproval	**Id:** 1 **Name:** MaterialRequest	ndTask

Figura 54 : Nós da definição do processo *MaterialRequest*

Tabela de Transitions:

Id	Name	From Node	To Node	Process Definition	Event
1		**Id:** 1 **Type:** ndFork	**Id:** 4 **Type:** ndAutoTask **Name:** SendAcceptNotification	**Id:** 1 **Name:** MaterialRequest	False
2		**Id:** 1 **Type:** ndFork	**Id:** 10 **Type:** ndAutoTask **Name:** PlaceOrder	**Id:** 1 **Name:** MaterialRequest	False
3		**Id:** 2 **Type:** ndAutoTask **Name:** SendDirectorRejectNotification	**Id:** 7 **Type:** ndEndProcess	**Id:** 1 **Name:** MaterialRequest	False
4		**Id:** 4 **Type:** ndAutoTask **Name:** SendAcceptNotification	**Id:** 8 **Type:** ndEndProcess	**Id:** 1 **Name:** MaterialRequest	False
5		**Id:** 5 **Type:** ndStartProcess	**Id:** 11 **Type:** ndTask **Name:** SubDirectorApproval	**Id:** 1 **Name:** MaterialRequest	False
6		**Id:** 6 **Type:** ndAutoTask **Name:** SendSubDirectorRejectNotification	**Id:** 3 **Type:** ndEndProcess	**Id:** 1 **Name:** MaterialRequest	False
7		**Id:** 10 **Type:** ndAutoTask **Name:** PlaceOrder	**Id:** 9 **Type:** ndEndProcess	**Id:** 1 **Name:** MaterialRequest	False
8	Reject	**Id:** 11 **Type:** ndTask **Name:** SubDirectorApproval	**Id:** 6 **Type:** ndAutoTask **Name:** SendSubDirectorRejectNotification	**Id:** 1 **Name:** MaterialRequest	True
9	Approve	**Id:** 11 **Type:** ndTask **Name:** SubDirectorApproval	**Id:** 12 **Type:** ndTask **Name:** DirectorApproval	**Id:** 1 **Name:** MaterialRequest	True
10	Approve	**Id:** 12 **Type:** ndTask **Name:** DirectorApproval	**Id:** 1 **Type:** ndFork	**Id:** 1 **Name:** MaterialRequest	True
11	Reject	**Id:** 12 **Type:** ndTask **Name:** DirectorApproval	**Id:** 2 **Type:** ndAutoTask **Name:** SendDirectorRejectNotification	**Id:** 1 **Name:** MaterialRequest	True

Figura 55 : Ligações entre nós da definição do processo *MaterialRequest*

3.2 Classe Gerada

```
using System;
using System.Collections;
using OutSystems.HubEdition.RuntimePlatform;
using OutSystems.HubEdition.RuntimePlatform.ProcessRuntime;

namespace ssMaterialRequest {

   public class Process_MaterialRequestDefinition: ProcessDefinition {

      private static Process_MaterialRequestDefinition _definition =
                                        new Process_MaterialRequestDefinition();

      private Process_MaterialRequestDefinition() {
             ID = 1;

             Nodes = new Hashtable();


             Node node3253 = new Fork(1);
             Nodes.Add(node3253.ID, node3253);
             Node node3971 = new Activity(2,
                    new ActivityPreparation(Activity_node3971_Preparation));
             Nodes.Add(node3971.ID, node3971);
             Node node2534 = new End(3);
             Nodes.Add(node2534.ID, node2534);
             Node node2578 = new Activity(4,
                    new ActivityPreparation(Activity_node2578_Preparation));
             Nodes.Add(node2578.ID, node2578);
             Node node2533 = new Start(5);
             Nodes.Add(node2533.ID, node2533);
             FirstNode = node2533;
             Node node3247 = new Activity(6,
                    new ActivityPreparation(Activity_node3247_Preparation));
             Nodes.Add(node3247.ID, node3247);
             Node node2568 = new End(7);
             Nodes.Add(node2568.ID, node2568);
             Node node3258 = new End(8);
             Nodes.Add(node3258.ID, node3258);
             Node node3975 = new End(9);
             Nodes.Add(node3975.ID, node3975);
             Node node3254 = new Activity(10,
                    new ActivityPreparation(Activity_node3254_Preparation));
             Nodes.Add(node3254.ID, node3254);
             Node node2553 = new Activity(11);
             Nodes.Add(node2553.ID, node2553);
             Node node2554 = new Activity(12);
             Nodes.Add(node2554.ID, node2554);


             new Transition(node3253, node2578);
             new Transition(node3253, node3254);
```

```
        new Transition(node3971, node3975);
        new Transition(node2578, node2534);
        new Transition(node2533, node2553);
        new Transition(node3247, node2568);
        new Transition(node3254, node3258);
        new Transition(node2553, node3247, new Event("Reject"));
        new Transition(node2553, node2554, new Event("Approve"));
        new Transition(node2554, node3253, new Event("Approve"));
        new Transition(node2554, node3971, new Event("Reject"));
}

/// <summary>
/// Action <code>Activity_node1150_Preparation</code> that represents the
/// Service Studio process_preparation <code>Preparation</code>
/// <p> Description: </p>
/// </summary>
public static string Activity_node3971_Preparation(int inParamProcessId,
                                                   int inParamTokenId) {
        HeContext heContext = null;

        // READ OF LOCALS VARS
        Hashtable vars = new Hashtable();
        vars.Add("MATERIALID", null);
        vars.Add("QUANTITY", null);
        vars.Add("TOTALPRICE", null);

        ProcessDefinition.ReadContext(inParamProcessId,
                                      "osusr_ujo_Material1", vars);
        int inParamMaterialId=(int) vars ["MATERIALID" ];
        int inParamQuantity=(int) vars ["QUANTITY" ];
        int inParamTotalPrice=(int) vars ["TOTALPRICE" ];
        // END READ OF LOCALS VARS

        ...

        // WRITE OF LOCALS VARS
        vars = new Hashtable();
        vars.Add("MATERIALID", inParamMaterialId);
        vars.Add("QUANTITY", inParamQuantity);
        vars.Add("TOTALPRICE", inParamTotalPric_u41 ?;

        ProcessDefinition.WriteContext(inParamProcessId,
                                       "osusr_ujo_Material1", vars);
        // END WRITE OF LOCALS VARS

        return null;
```

```
}

/// <summary>
/// Action <code>Activity_node2578_Preparation</code> that represents the
/// Service Studio process_preparation <code>Preparation</code>
/// <p> Description: </p>
/// </summary>
public static string Activity_node2578_Preparation(int inParamProcessId,
                                                   int inParamTokenId) {
       HeContext heContext = null;

       ...

       return null;
}

/// <summary>
/// Action <code>Activity_node3247_Preparation</code> that represents the
/// Service Studio process_preparation <code>Preparation</code>
/// <p> Description: </p>
/// </summary>
public static string Activity_node3247_Preparation(int inParamProcessId,
                                                   int inParamTokenId) {
       HeContext heContext = null;

       ...

       return null;
}

/// <summary>
/// Action <code>Activity_node3254_Preparation</code> that represents the
/// Service Studio process_preparation <code>Preparation</code>
/// <p> Description: </p>
/// </summary>
public static string Activity_node3254_Preparation(int inParamProcessId,
                                                   int inParamTokenId) {
       HeContext heContext = null;

       ...

       return null;
}

public static void RegisterDefinition() {
       ProcessEngine.ProcessDefinitions.Add(_definition.ID, _definition);
}
```

```
        public static int CreateInstance() {
                return ProcessRuntime.CreateProcess(_definition);
        }

        public static int CreateSubInstance(int inParamTokenId, int Node) {
                return ProcessRuntime.CreateProcess(_definition, inParamTokenId,
                                                    Node);
        }
    }
}
```

4. Glossário

- **1-CP** *(1-Click Publish)* **–** Processo de publicação de uma aplicação OutSystems com apenas um *click*. Durante este processo a aplicação é verificada, guardada, é feito o *upload* para o *HubServer*, é compilada (inclui a fase de geração de código) e é instalada no servidor, ficando pronta para execução.
- ***Agile*** **–** Nova metodologia para o desenvolvimento de software. Esta metodologia promove a comunicação cara-a-cara em detrimento de documentos escritos, e vê o software em funcionamento como a principal medida de progresso.
- **Definição de Processo –** Termo utilizado para designar o resultado da modelação de um processo. É o *Flow* resultante da modelação que detalha todas as actividades e dependências entre elas, que espelham um processo.
- **Ecrã –** Termo utilizado para designar uma interface de utilizador desenvolvida no modelador da plataforma OutSystems. Embora possam ser desenvolvidos vários tipos de ecrã com esta plataforma, neste trabalho este termo refere-se a páginas *Web*.
- ***eSpace*** **-** Um *eSpace* OutSystems é uma aplicação ou parte de uma aplicação desenvolvida na plataforma OutSystems. Estas aplicações são normalmente compostas por três camadas, a interface com o utilizador, a lógica de negócio e o modelo de dados.
- **Evento –** Elemento da linguagem de modelação que indica que durante a execução do processo a transição entre dois nós só é executada quando o processo for notificado do exterior que pode retomar a sua execução.
- **Gerador de Código –** Componente interno ao *HubServer* que processa o *output* do *Service Studio* (componente de modelação) e produz os *inputs* necessários para a execução. Este componente gera o código para as aplicações desenvolvidas visualmente no modelador.
- **Gestão de Processos de Negócio –** Disciplina que estuda o impacto e benefícios dos processos de negócio nas organizações. Esta disciplina procura organizar operações e consolidar organizações através dos processos de negócio.
- ***HubEdition*** **–** Nome utilizado para designar a plataforma OutSystems com todos os seus componentes, o *HubServer*, o *Service Studio*, o *Integration Studio* e o *Service Center*.
- ***HubServer*** **–** Componente da plataforma OutSystems que orquestra toda a execução, instalação e gestão das aplicações.
- **Instância de Processo –** Termo utilizado para designar a execução de uma definição de processo. Visto que uma definição de processo pode ser executada várias vezes, pode dar origem a várias instâncias de processo.
- ***Integration Studio*** **–** Componente da plataforma OutSystems que permite desenvolver elementos de integração com outras aplicações ou mesmo desenvolver elementos que não possam ser desenvolvidos directamente no componente de modelação. Este componente produz elementos que podem ser utilizados visualmente no componente de modelação.

- **Motor de Execução –** Novo componente interno do *HubServer* que orquestra toda a execução de processos negócio na plataforma OutSystems. Este está dependente de um modelo de dados onde é guardada a informação referente aos processos.
- **Nó –** Termo utilizado para definir uma primitiva da nova linguagem de modelação de processos de negócio da plataforma OutSystems.
- **OML** *(OutSystems Markup Language)* – É o formato em que são guardadas as aplicações desenvolvidas no componente de modelação da plataforma OutSystems. Este termo é também utilizado para referir o ficheiro produzido pelo componente de modelação e que armazena toda a aplicação desenvolvida.
- **OutSystems –** Organização de software e serviços criada em 2001. O negócio desta organização baseia-se na plataforma que criaram, que com a nova metodologia promovida permite obter óptimos resultados no desenvolvimento de software.
- ***Preparation* –** Refere a lógica de negócio que algumas primitivas da nova linguagem de modelação de processos da plataforma, podem possuir. Esta lógica de negócio é executada antes da semântica das primitivas.
- ***ProcessAPI* –** Conjunto de funções que o componente de modelação disponibiliza para que o modelador possa criar novas instâncias de processo e possa interagir com as instâncias em execução.
- ***ProcessFlow* –** Termo pelo qual as definições de processo (actividades e dependências que definem um processo), são referidas no componente de modelação da plataforma OutSystems.
- **Processo de Negócio** – É o conjunto completo e dinâmico de actividades colaborativas que produzem valor. Os processos de negócio representam a forma como o trabalho é realizado e podem existir independentemente de qualquer tecnologia.
- **SCRUM –** É um método ágil para gestão de projectos que permite atingir uma elevada produtividade. Este assume que o desenvolvimento é complicado e imprevisível tratando-o como uma caixa negra controlada e não como um processo completamento definido. Este termo é também utilizado para designar reuniões diárias de pé, com duração de poucos minutos com o intuito de relatar progressos e dificuldades actuais no desenvolvimento. Estas reuniões são promovidas pelo método SCRUM.
- ***Service Center* –** Componente da plataforma OutSystems que monitoriza e gere todas as aplicações que se estão a executar sobre a plataforma.
- ***Service Studio* –** Componente da plataforma OutSystems onde é realizada toda a fase de desenvolvimento de aplicações. O seu ambiente *drag'n'drop* permite um desenvolvimento bastante mais rápido e fácil que o tradicional, oferecendo meios para criar visualmente, interfaces, lógica de negócio, repositórios de informação, *WebServices*, etc.
- **SQL** *(Strutured Query Language)* **–** Linguagem desenvolvida pela IBM que permite retirar e actualizar informação estruturada de uma base de dados.
- **TI** (Tecnologias da Informação) **–** O ramo da engenharia responsável pelo estudo de computadores e comunicações como meio de recolha, armazenamento e processamento de informação.

- ***Token* de Processo –** Termo utilizado para designar um fluxo de execução numa instância de processo. Visto que uma instância de processo pode ter vários fluxos de execução, esta possui sempre um ou mais *Tokens* de processo.
- **Transição –** Termo utilizado para definir uma ligação entre duas primitivas na nova linguagem de modelação de processos de negócio da plataforma OutSystems.
- ***WebService* –** Colecção de protocolos e standards usados para troca de informação entre aplicações. Permite que aplicações desenvolvidas nas mais variadas linguagens de programação e a executarem-se sobre a mais variada gama de plataformas, possam interagir e trocar informação através de redes de computadores.
- **XML** *(eXtensible Markup Language)* **–** Standard para a criação de *markup languages* que descrevem a estrutura da informação. O XML permite aos autores definirem as suas próprias *tags*. Esta possui uma definição formal desenvolvida pelo *World Wide Web Consortium*.