INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

DEPARTAMENTO DE ENGENHARIA DE ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

SISTEMAS DE INFORMAÇÃO

Operação em Ambiente Móvel Baseado em Modelo de Mercado

PROJECTO PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Ricardo José Gomes Silva
(Licenciado em Engenharia Informática e de Computadores)

Orientador:

Professor Doutor Paulo Manuel Trigo Cândido da Silva

Juri:

Presidente:

Mestre Vitor Jesus Sousa de Almeida

Vogais:

Doutor Fernando Jorge Ferreira Lopes

Professor Doutor Paulo Manuel Trigo Cândido da Silva

SETEMBRO DE 2013

Resumo

A evolução das tecnologias associadas à Internet (a que se assiste continuamente) está a modificar a forma como a informação é produzida e armazenada permitindo o seu acesso em qualquer local. As empresas pretendem que os colaboradores, que realizam trabalho no terreno, enviem a informação das suas tarefas de forma rápida de modo a ser registada nos sistemas informáticos. O pressuposto essencial para a realização dessas operações é a existência de conectividade. Apesar do acesso sem-fio à Internet ser amplamente utilizado, existem situações onde os utilizadores estão privados dessa ligação. Este projeto apresenta modelos para o desenvolvimento de aplicações *Web-Offline*.

O *Web-Offline* é um conceito enquadrado nas tecnologias de Internet que permite o acesso a um *Website* através do *Browser* sem a necessidade de uma ligação permanente à rede global. Neste sentido foi implementada uma aplicação com estas características que permite aos utilizadores realizarem as suas tarefas sem uma dependência contínua de conectividade. As tarefas são disponibilizadas aos utilizadores através de *Backoffice*. As escolhas das tarefas são negociadas, pelos utilizadores entre si, através de um mecanismo de leilão. A aplicação implementada foi testada por diversos utilizadores de forma a avaliar a sua usabilidade. Os resultados obtidos demostraram uma recetividade positiva por parte dos utilizadores.

Palavras - chave

Web-Offline

Conectividade

Leilão

Abstract

Currently, the evolution of new Internet related technologies (which is continually increasing) is constantly changing the way information is produced and stored allowing easy access worldwide. Workers are encouraged by companies to send the information regarding its tasks quickly and efficiently, in order to be properly stored in the computer system. One of the major and essential prerequisite for the accomplishment of these operations is the existence of connectivity. However, despite the extensive access to wireless Internet, in some situations users are deprived of this connection. The main goal of this project is to present alternative models for the development of applications offline (*Web-Offline).*

The *Web-Offline* is a new concept in the range of Internet technologies which allow easy access to a *Website* through the *Browser* without the need for a permanent connection to the global network. Therefore, we implemented an application available via *Backoffice,* which allows users to perform their tasks without the need for continued connectivity. Tasks selections are negotiated between users through an auction mechanism. The application has been tested and implemented by different users in order to evaluate its usability. The results demonstrated a very positive reception by the users.

Palavras - chave

Web-Offline

Connectivity

Auction

Agradecimentos

Este espaço é dedicado àqueles que deram a sua contribuição para que esta dissertação fosse realizada. A todos eles deixo aqui o meu sincero agradecimento.

Em primeiro lugar, quero agradecer ao meu orientador de dissertação, Professor Doutor Paulo Trigo, pela disponibilidade, apoio, incentivo e acompanhamento ao longo deste trabalho.

Quero também agradecer à minha mulher pelo modo como me aturou, pelo modo como sempre me apoiou e acompanhou ao longo deste difícil trajeto. Quero especialmente pedir desculpa à minha filha pela minha ausência em momentos importantes.

Aos meus pais, José e Lucília, pela forma como me incutiram a alegria de viver, e pela confiança que sempre depositaram em mim.

Finalmente ao Nuno, meu colega de mestrado que sempre me incentivou, mesmo quando estava mais desanimado.

Obrigado!

Glossário de termos e abreviaturas

API- (*Application Programming Interface*) Conjunto de funções de *Software* para utilização programática

***BackOffice* -** *Software* que suporta a atividade empresarial que não está visível ao utilizador final

DOM- (*Document Object Model*) Especificação para alteração dinâmica de um documento *HTML*

HTTP- (*Hypertext Transfer Protocol)* Protocolo de hipertexto utilizado na *World Wide Web*

***JSON* -** (*JavaScript Object Notation*) Formato para troca de dados computacionais com notação baseada em objetos *Javascript*

Maschups- *Website* que utiliza várias fontes de dados na disponibilização do seu serviço

***Plugin* -** Módulo de *Software* para adicionar funções a programas existentes

***Runtime* -** Ambiente de execução realizado por máquinas virtuais para gerir aplicações

SOAP - (*Simple Object Access Protocol*) Protocolo para troca de mensagens baseado em *XML*

SQLite- Base de dados relacional constituída por um conjunto limitado de funcionalidades

***WPF* -** (*Windows Presentation Foundation*) Componente da plataforma *.Net Framework* para o desenvolvimento de aplicações gráficas *Windows*

ACID – (*Atomicity, Consistency,* Isolation *e Durability*) Conceito para caracterizar um transação

URL- (*Uniform Resource Locator*) Identificador de um recurso disponível numa rede

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

No último ano participei num projeto a nível profissional que tinha como objetivo o controlo de qualidade a técnicos que prestam serviço à minha empresa. A realização dessa operação pressupunha a existência de conectividade com o servidor central que nem sempre era possível. Foi desenvolvida uma aplicação móvel nativa que permitia aos utilizadores avaliarem o trabalho dos técnicos. Na ausência de ligação com o sistema informático central os dados eram gravados localmente no dispositivo. Após restabelecida a ligação os dados eram posteriormente enviados.

Com o decorrer do projeto identificou-se algumas debilidades na solução implementada nomeadamente

- Dificuldades na distribuição de pessoas para avaliação dos técnicos
- Dificuldades na manutenção de novas versões da aplicação

Para a afetação de uma pessoa era necessário considerar o local da auditoria e o tipo de auditoria. A impossibilidade da realização de uma avaliação por parte de uma pessoa implicava o ajuste das restantes.

A componente de negócio era bastante volátil provocando constantes alterações à aplicação, o que obrigava o utilizador a descarregar novas versões. Este procedimento nem sempre era bem sucedido tornando a manutenção da aplicação difícil e complexa, condicionando o escalonamento da aplicação a muitos utilizadores.

Este projeto tem como objetivo desenvolver uma aplicação *Web* que suporte mobilidade e que se mantenha disponível mesmo na ausência de uma ligação à Internet permitindo ao utilizador a execução das suas tarefas. As tarefas a realizar são criadas numa aplicação (também desenvolvida neste projeto) que executa em *BackOffice*.

A atribuição de tarefas será realizada através de um mecanismo de leilão onde cada utilizador escolhe (lícita) as tarefas que lhe são mais convenientes. A adoção de um modelo de leilão simplifica todo o processo de atribuição de tarefas e permite usar o

modelo de mercado para encontrar os pontos de equilíbrio entre os valores de utilidade das tarefas e o desempenho dos utilizadores na realização dessas tarefas.

A atribuição das tarefas a realizar por parte dos utilizadores segue os seguintes passos:

1- Oferta das tarefas a realizar (várias vezes ao dia);

2- Licitação de propostas por cada utilizador;

3- Afetação de tarefas a utilizadores.

Regressar ao passo 1 caso existam tarefas por atribuir.

Na ausência de conectividade por parte do utilizador para submeter a licitação de determinada tarefa o sistema garante a persistência local dessa tarefa. Caso o utilizador consiga restabelecer a conectividade antes do final do leilão a sua licitação será automaticamente enviada e considerada para a processo de afetação.

As tarefas poderão ter diferentes características (e.g., tema, dificuldade, estrutura) e a cada tarefa é atribuído um valor de referência para licitação (indicador da utilidade esperada da tarefa). Essa caraterização será realizada na criação da tarefa em *BackOffice*. É nesse momento que é definido o valor de referência para a licitação da tarefa.

Cada utilizador tem um número de pontos (inicialmente todos com o mesmo valor). Esse número de pontos será aumentado à medida que vai realizando as tarefas. A pontuação obtida é avaliada segundo a eficácia e a eficiência. Caso a tarefa não seja realizada são retirados os pontos correspondentes ao valor licitado na atribuição da tarefa.

Outro objetivo deste projeto é a criação de um motor para criação de tarefas que possa ser generalizado para diferentes contextos. Pretende-se neste projeto generalizar o conceito de tarefa a outros domínios de aplicação. Genericamente cada tarefa é um conjunto de atividades a executar (pelo utilizador) de modo a registar, em formato do tipo questionário, o resultado dessa execução.

O resultado de cada atividade realizada pelo utilizador pode desencadear atividades diferentes que são apresentadas na aplicação dinamicamente. As atividades estão relacionadas entre si segundo a sua execução, sob forma de uma árvore, em que os elementos são as atividades e os nós as respostas. A alteração de uma atividade pode dar origem a uma árvore diferente daquela que existia antes dessa alteração.

De forma a dar ao utilizador alguma flexibilidade na realização da tarefa foi implementado na aplicação a possibilidade de guardar diferentes árvores de navegação. Deste modo, sempre que é desfeita uma árvore de atividades é guardada em memória essa estrutura. Caso o utilizador por engano modifique uma atividade dando origem a uma nova árvore, pode refazer a alteração repondo a última árvore desfeita.

O resultado da atividade pode ser registado por resposta fixa ou editável. O motor implementado para criação de tarefas suporta um número variável de respostas fixas. O conteúdo das respostas é também configurável.

Neste projeto o motor implementado foi utilizado para dois domínios de aplicação distintos. Um domínio é a auditoria ao trabalho dos técnicos de modo a avaliar se os pressupostos estipulados estão a ser cumpridos. O outro domínio de aplicação tem um carácter pedagógico com a criação de várias perguntas de um determinado conteúdo programático.

2. Trabalho Relacionado

De forma a conhecer soluções existentes que utilizam o mecanismo de negociação baseado em leilões foi analisado o sistema *eBay*. Neste capítulo são apresentados e caraterizados os vários tipos de leilões suportados por esta plataforma.

São também descritas duas soluções tidas como candidatas ao suporte *Offline*. Uma solução apresentada baseia-se na incorporação de plataformas proprietárias enquanto a outra tem por base apenas a utilização de *Browser* como plataforma de execução. Será realizado um paralelismo entre estas soluções e justificada a opção tomada. São apresentados também, alguns exemplos de *Web Applications* que disponibilizam funcionalidades *Offline*.

Será ainda apresentado o protocolo utilizado na comunicação cliente-servidor e o padrão de desenho utilizado no projeto, que visa separar a camada de dados com a interface gráfica.

2.1. *eBay* – Tipo de Leilões Existentes

O *eBay* é o site mais popular de *shopping* na Internet onde é possível a compra e venda de bens.

A plataforma *eBay* suporta quatro tipos de leilão: (1)

- *Normal Auctions* (*English Auction*)
- *Reserve Auction*
- *Multiple Item Auctions*
- *Fixed Price* (*Buy it Now*)

O tipo leilão "*Normal Auctions*" é considerado o leilão existente mais antigo, conhecido pelo leilão ascendente em que o preço do bem é aumentado sucessivamente até que reste um participante disposto a adquiri-lo.

No tipo leilão "Reserve *Auction*" o licitador indica o valor mínimo para a transação do bem. Se esse valor não for alcançado o bem não será negociado. Este tipo de leilão garante que o vendedor recebe uma fração significativa do valor do bem mesmo quando a competição é fraca.

No tipo leilão "*Multiple item Auctions*" são disponibilizados para leilão várias unidades de um determinado bem. O comprador que licitar o valor mais alto (por unidade) ficará com os bens pelo valor mais baixo licitado. Este tipo de leilão é utilizado normalmente em conjunto com o "*Reserve Auction*" de forma a evitar a venda dos bens por valores muito baixos.

No tipo de leilão "*Fixed Price*" é colocado o valor fixo para o bem, o primeiro comprador a licitar esse valor fica com o bem.

Os leilões apresentados são leilões abertos uma vez que as propostas são públicas e conhecidas pelos participantes. Existem no entanto leilões em que a proposta de cada participante é apenas conhecida no final do leilão. Estes tipos de leilões são denominados leilões fechados.

Um exemplo deste tipo de leilão são os concursos de projetos para empresas. Cada empresa que concorre a estes concursos indica o valor que quer receber para a implementação do projeto. As diferentes propostas mantem-se privadas até ao final do concurso.

2.2. Soluções Offline

O acesso à plataforma *eBay* é realizado pelo cliente *HTTP*, tipicamente um *Browser*. Imaginemos agora um cenário em que o utilizador no exato momento que está a realizar a licitação fica sem conectividade. Neste cenário, o utilizador fica sem saber se a sua licitação foi realmente efetuada.

Uma das lacunas apontadas às aplicações *Web* está relacionada com a dependência de conectividade entre o cliente e o servidor. Acresce a esta lacuna a forte limitação de persistência de dados de forma permanente. Empresas como *Adobe* e *Microsoft* e *Google* lançaram soluções de forma a contornar essas limitações. Essas soluções permitem o funcionamento das aplicações em modo *Offline* através da utilização das suas plataformas:

- *Adobe Integrated Runtime* (*AIR*)
- *Mozila Prism*
- *Silverlight Out-Of-Browser*

O *Adobe AIR* (2) é um *Runtime* que permite combinar diversas tecnologias para a implementação de *Rich Internet Applications* (3). Esta plataforma permite que as aplicações *Web* sejam executadas fora do *Browser* utilizando tecnologias usuais da Internet *(*e.g*., HTML, CSS, JavaScript, Flash, Flex).*

O *AIR* permite utilizar como plataforma de execução o *Browser* ou o próprio *Runtime*. No ambiente executado no *Browser* o modelo de segurança é restrito nomeadamente no acesso ao disco rígido do utilizador. Quando é executado diretamente no *Runtime* o acesso é mais alargado permitindo o acesso a *APIs* específicas possibilitando escrita e leitura em disco.

O *Adobe AIR* consegue manter as capacidades e tecnologias suportadas pelos *Browsers* estendendo-as ao ambiente *Desktop*, permitindo um comportamento semelhante às aplicações nativas.

O *Mozila PRISM* (4) foi desenhado para ser independente da plataforma onde é utilizado tornando as aplicações facilmente portáteis entre os sistemas operativos. Para isso utiliza uma linguagem declarativa *XUL* (*eXtensible User Interface Language*) que oferece um nível de abstração de modo a que o mesmo código seja executado por diferentes plataformas.

O *XUL* é uma linguagem declarativa baseada em *XML* desenvolvida pelo *Mozila Foundation* que permite a construção de aplicações multiplataforma. É uma linguagem orientada ao componente gráfico, como janelas, botões e etiquetas, em vez de páginas e tabelas como a linguagem *DHTML* (*Dynamic HyperText Transfer Protocol*).

A interface gráfica de todos produtos principais do *Mozila* (e.g., *Browser*) é implementada utilizando o *XUL* com um código único base que suporta todas as plataformas *Mozila*.

O *Mozila Prism* é um interpretador de *XUL* (também designado *XULRunner*) com uma interface diferente dos *Browsers* típicos. É baseado num conceito designado *Site Specific Browser* (SSB). Um *SSB* é uma aplicação (com um *Browser* embebido) que tem como objetivo executar *Web Applications*. O *Prism* pretende diminuir as diferenças

que existem entre as *Desktop Applications* e as *Web Applications* com vista à melhoraria da experiência de utilização em aplicações *Web*.

O *Silverlight Out-Of-browser (OOB)* (5) é uma tecnologia que tal como o *Prism* utiliza uma linguagem declarativa na construção de interfaces gráficas. O *XAML* (*eXtensible Application Markup Language*) foi inicialmente implementado pela *Microsoft* para o desenvolvimento de aplicações gráficas *Windows (WPF)*.

Posteriormente a *Microsoft* lançou um produto o *Silverlight* que permite o desenvolvimento de aplicações *RIA*. O *Silverlight* é um *Cross-Plataform-Plugin* que inclui um subconjunto das funcionalidades *WPF,* utilizando igualmente a linguagem *XAML* para a construção visual das aplicações *Web. O Silverlight* é compatível com a maior parte dos *Browsers* e é instalado através de um *Plugin*.

Com o lançamento do *Silverlight 3.0* a *Microsoft* lançou o *Silverlight Out-Of-browser* que permite executar qualquer aplicação *Silverlight* num processo isolado do *Browser*. Assim, a execução é realizada como se de uma aplicação nativa se tratasse permitindo o acesso ao sistema de ficheiros da máquina do utilizador.

Vamos agora abordar outro tipo de solução que utiliza apenas o *Browser* como plataforma de execução.

- *Google Gears*
- *HTML5*

O *Google Gears* (6) é uma extensão às funcionalidades do *Browser* implementado pelo *Google* que permite a execução de aplicações *Web* mesmo sem conexão com o servidor. É disponibilizado através da instalação de um *plugin* e é compatível com a maioria dos *Browsers*.

E constituído por três componentes principais

- *LocalServer*: permite armazenamento local das páginas *Web*
- *Database*: permite armazenamento de dados locais
- *WorkerPool*: permite a execução de operações de forma assíncrona

O módulo *Local Server* é um *cache* onde são persistidas no disco rígido do utilizador as páginas existentes na aplicação *Web*. Quando está em modo ativo, os pedidos são realizados nesse *cache* que responde com a página solicitada. Para estar ativo a propriedade *enable* deste *cache* terá de estar com valor *true*. Uma vez ativado todos os pedidos serão redirecionados para o *Local Server*, quer exista ou não conectividade.

O módulo *Database* permite a persistência dos dados localmente. Os dados são guardados no disco rígido através de uma base de dados relacional *SQLite*.

O módulo *WorkerPool* tem como objetivo oferecer uma boa experiência ao utilizador com a execução assíncrona de operações que envolvam computação pesada. Deste modo, a interface gráfica do utilizador não é bloqueada.

O *HTML 5* (7) tem uma infraestrutura semelhante ao *Google Gears*. Possui também um *cache* local (*AppCache*) que tal como o *Google Gears*, é utilizado independentemente do estado da conectividade.

O *AppCache* permite indicar explicitamente ao *Browser* quais os recursos necessários (*HTML*, *Javascript*, *CSS*, imagens) que deverão ser guardados de modo persistente em *cache*, de forma a permitir o funcionamento *Offline*.

O *AppCache* poderá também ser utilizado em modo *Online* para melhorar o desempenho da aplicação. Assim o *Browser* acede aos recursos diretamente no *AppCache* evitando o *overhead* do acesso ao servidor. Aliás, esse é o funcionamento por omissão, pois se a versão (informação presente no manifesto) carregada é a mesma que está na *AppCache* o *Browser* utiliza sempre os recursos em *cache*. Existe no entanto a possibilidade de modificar a versão do manifesto de forma programática forçando o *Browser* a aceder aos recursos do servidor.

Para utilizar o *AppCache* é necessário criar um ficheiro (o manifesto) e indicar na página *HTML* o local onde se encontra esse ficheiro. O ficheiro manifesto está dividido em três secções: *Cache Section*, *Network Section* e *Fallback Sections*. Na secção *Cache Section* é indicada a lista de recursos sobre a forma de *url* que serão explicitamente armazenados em *cache* após o primeiro carregamento. Na secção *Network* são listados os ficheiros que nunca deverão estar em *cache*. Em caso de existirem, o *Browser* lança

um erro. Na secção *Fallback* é indicado qual o ficheiro a carregar caso o *Browser* não encontre a página no *AppCache*.

A utilização do *AppCache* é uma solução necessária para o suporte de aplicações *Offline*, podendo ser utilizada também de forma a maximizar o desempenho, contendo algumas limitações. Possui uma *API* muito limitada não permitindo por exemplo, saber se determinado recurso está em *cache* ou não, ou limpar o *cache* de forma programática.

Outra funcionalidade oferecida pelo *HTML5* necessária à implementação de aplicações *Offline* é o armazenamento de dados no *Browser*. O *HTML5* disponibiliza quatro formas de armazenamento: *Web Storages*, *Web SQL Databases*, *IndexDB* e *File API*.

O *Web Storage* é suportado pela maioria dos *Browsers*, tendo um funcionamento muito simples baseado em pares chave/valor. A chave é identificada sobre forma de *String* e o valor é qualquer tipo *Javascript*, incluindo os tipos primitivos e objetos. O *Web Storage* pode ser ainda dividido em dois grupos: o *Session Storage* e o *Local Storage*. Utilizando o *Session Storage* os dados estão disponíveis apenas para a sessão (janela) que criou o dado, não existindo acesso a esse dado noutra sessão (janela). No entanto estes dados estão apenas disponíveis enquanto a sessão existir. No *Local Storage*, por outro lado, a informação é partilhada por todas as sessões (janelas) ficando a informação disponível mesmo que o *Browser* feche.

O *Web SQL Database* é uma forma de armazenamento com base no modelo relacional. Este tipo de armazenamento não é suportado por todos os *Browsers* caracterizando-se como um tipo de armazenamento um pouco rígido e bastante complexo. Possui no entanto um bom nível de desempenho nas leituras sempre que indexado pelas chaves primárias.

O tipo de armazenamento *IndexDB* tira partido do melhor dos dois mundos entre o *Web Storage* e o *Web SQL Database*. Não possui uma estrutura rígida como o *Web Storage* e aproveita o desempenho das leituras do *Web SQL Database*.

O tipo de armazenamento *File Api* é um tipo de armazenamento utilizado para recursos pesados (áudio, vídeo, imagens) sendo implementado apenas pelo *Browser Google Chrome*.

2.2.1. Escolha da tecnologia

Após a análise das tecnologias apresentadas foi realizada uma análise comparativa de modo a perceber qual a tecnologia mais adequada ao projeto. A tabela 2.1 sintetiza as diferenças entre os dois tipos soluções estudadas.

	Execução no Browser	Execução em Multiplataforma
Distribuição	As aplicações são disponibilizadas através de um *Web Server*	É necessário uma prévia instalação de plataformas ou *runtime* nas máquinas dos utilizadores para execução das aplicações
Segurança	Os dados são armazenados numa infraestrutura restrita disponilizada pelo *Browser*	Permite o acesso ao disco rígido do utilizador como forma de persistência
Linguagens de Programação	Linguagem suportada pelo *Browser – Javascript*	Linguagens específicas das plataformas
Atualizações	As aplicações são atualizadas no servidor	Poderá ser necessário realizar atualizações à plataforma utilizada para a execução das aplicações
Interface Gráfica	Utilização de controlos próprios dos *Browsers*	As aplicações são visualizadas em janelas próprias

Tabela 2.1 – Tabela comparativa de soluções *Offline* estudadas

As aplicações multiplataforma fornecem a capacidade para execução *Offline.* Contudo a sua origem não está relacionada exclusivamente com esta funcionalidade. O principal objetivo destas plataformas é reduzir a diferença entre as aplicações *Desktop* e as aplicações *Web* explorando novos modelos de usabilidade.

A sua utilização implica uma dependência tecnológica com os fornecedores. Os erros existentes nestas plataformas apenas poderão ser corrigidos pelos próprios fornecedores.

As soluções "execução no *Browser*" permitem uma mais fácil manutenção das aplicações. As aplicações são atualizadas num único sítio (servidor *Web*) ficando imediatamente disponíveis a todos os utilizadores.

A tecnologia escolhida para a implementação deste projeto foi o *HTML5*. A distribuição da aplicação ao utilizador não requer qualquer tipo de instalação nas máquinas dos clientes.

O *HTML5* fornece uma especificação, ao contrário do *Google Gears* que disponibiliza apenas uma implementação. O *HTML5* permite a implementação das funcionalidades *Offline* pelos mais diversos *Browsers*.

Com o aparecimento do *HTML5* o *Google* descontinuou o projeto *Google Gears.* A sua utilização continua a ser pública mas não é mantida.

2.2.2. *Web Applications* com funcionalidades *Offline*

Nesta secção apresentam-se algumas *Web Applications* que disponibilizam funcionalidades *Offline*:

- *Ebay Desktop*
- *Google Reader*
- *Google Docs*

O *Ebay Desktop* permite a operação de compra e venda de bens sem que exista conectividade com os servidores centrais do *eBay*. Esta solução foi desenvolvida em parceria com o *Adobe* utilizando a plataforma *AIR*.

O *Google Reader* permitia gerir a subscrição de vários sites que disponibilizavam conteúdos no formato *RSS*. Foi a primeira *Web Application* do *Google* com uma versão *Offline* publicamente acessível.

Esta aplicação implementava o modo "utilizador decide" dando a possibilidade ao utilizador de alternar entre os modos *Online* e *Offline*. Na comutação para o estado *Offline* esta funcionalidade permitia a transferência para o cliente de cerca de dois mil itens. Inicialmente foi realizada através do *plugin Google Geeks*, mas com a sua descontinuidade, foi implementada em *HTML5*. Em junho de 2013 o Google anunciou o fim desta aplicação.

O *Google Docs* é uma *Web Application* que permite a criação e edição de documentos de texto, folhas de cálculo de apresentações. Tal como *Google Reader* foi o primeiro implementado com o *Google Gears* e posteriormente migrado para *HTML5*

2.3. Protocolo de Comunicação

Uma tecnologia muito utilizada e aceite pela indústria na comunicação cliente-servidor são os *Web Services*. Os *Web Services* utilizam o protocolo *SOAP* como forma de comunicação. Neste protocolo a informação é transmitida através de um documento *XML* onde estão incluídas diversas especificações como o *WSDL,* de forma a garantir a interoperabilidade entre sistemas.

Com o aparecimento de aplicações *Maschups* surgiu a necessidade do mesmo serviço disponibilizar informação em diferentes formatos.

O *REST* (*Representational State Transfer*) (8) apareceu de forma a responder a este novo paradigma de representação da informação em diversos formatos. Este estilo de arquitetura define qualquer interface que transmita dados sobre *HTTP* sem a necessidade de qualquer camada adicional de mensagens (e.g., *SOAP*).

No paradigma *REST* qualquer serviço ou operação é identificado segundo um *URI* único. Para invocar qualquer serviço utilizam-se os verbos da interface uniforme *HTTP*

- *GET*: devolve dados de um recurso
- *PUT*: atualiza dados de um recurso
- *POST*: adiciona dados a um recurso
- *DELETE*: remove dados de um recurso

Para este projeto, dado as especificidades das tecnologias utilizadas (baseadas em *Javascript*), o formato de dados na comunicação é *JSON*. Desta forma optou-se pela utilização da arquitetura *REST* como protocolo de comunicação.

2.4. Padrão MVVM

Com o aumento da complexidade das *Web Applications* surgiu a necessidade da divisão das aplicações em camadas lógicas de modo a melhorar o processo de desenvolvimento e a capacidade de manutenção da aplicação. Tipicamente a arquitetura de uma aplicação *Web* está dividida em três camadas lógicas

- Interface
- Lógica de negócio
- Camada de dados

O *MVVM* (*Model-View-View-Model)* (9) é um padrão de desenho que visa estabelecer uma clara separação entre a interface gráfica e os modelos de dados. Segundo o *MVVM* a responsabilidade da *View* está apenas relacionada com a componente visual que é mostrada ao utilizador. O *Model* representa a camada de dados e o *ViewModel* estabelece a ligação entre *View* e o *Model*.

3. Modelo Proposto

Neste capítulo são apresentados os modelos que constituíram a base deste trabalho. A figura 3.1 apresenta os três grandes módulos que constituem a solução implementada.

Figura 3.1- Módulos da solução

A *Framework* é o módulo de suporta o modelo de desenvolvimento orientando o programador a uma escrita de código sistemática e organizada. Internamente é constituída por dois componentes

- Máquina de estados
- Contentor de modelos

A máquina de estados permite incutir no programador um raciocínio padronizado baseado na divisão de responsabilidades funcionais em diversos estados, promovendo a modularidade na implementação.

O contentor de modelos é o módulo responsável pelo armazenamento de objetos que representam dados. Estes objetos implementam o padrão *MVVM* com atualização automática dos dados na interface gráfica. Os dados podem ser persistidos no servidor ou localmente.

O modelo de mercado é responsável para afetação de recursos (tarefas) aos utilizadores, concretizado no mecanismo de leilão. Este modelo pretende oferecer aos utilizadores (inseridos num contexto móvel) alguma flexibilidade na escolha das tarefas mais apropriadas em cada momento.

O motor de criação de tarefas permite a criação de tarefas em *BackOffice* que serão disponibilizadas aos utilizadores em leilão.

3.1. Arquiteturas *Web*

Com o aparecimento das linguagens *server-side* (10) (e.g., *PHP*,*ASP*) as páginas *Web* tornaram-se aplicações (*Web Applications*), em que a componente dinâmica está toda no servidor. Estas aplicações respondem aos *inputs* do utilizador (através do *Browser*) com a geração de conteúdo dinâmico codificado numa linguagem (e.g., *HTML,XML*).

O aparecimento da *Web 2.0* pretendeu melhorar a experiência de utilização das *Web Applications*. Surgiu um novo conceito *AJAX* (*Asynchronous Javascript and XML*) (11) que introduziu a capacidade assíncrona de envio de pedidos e de recepção de respostas. O *AJAX* utiliza um conjunto de tecnologias no lado do cliente (e.g., *Javascript*) que são responsáveis pela comunicação com o servidor e por atualizar a interface com o resultado desta comunicação.

Emerge atualmente um novo paradigma de aplicações (*Web-Offline)* que para além de manter todas as características das *Web Applications*, mantêm-se disponíveis mesmo na ausência de conectividade. Nas aplicações *Web-Offline* a lógica funcional está centrada no lado do cliente e a comunicação com o servidor é apenas realizada para acesso a dados. Nesta secção são ilustrados exemplos do comportamento de cada arquitetura num contexto *Offline*.

Nas aplicações *Server Side Web Applications* quando o utilizador pressiona um botão, ou uma hiperligação (*link*) é efetuado um pedido *HTTP* (*POST* ou *GET*) ao servidor que realiza o processamento adequado a esse pedido e responde com uma nova página *HTML*. A figura 3.2 exemplifica a utilização deste modelo.

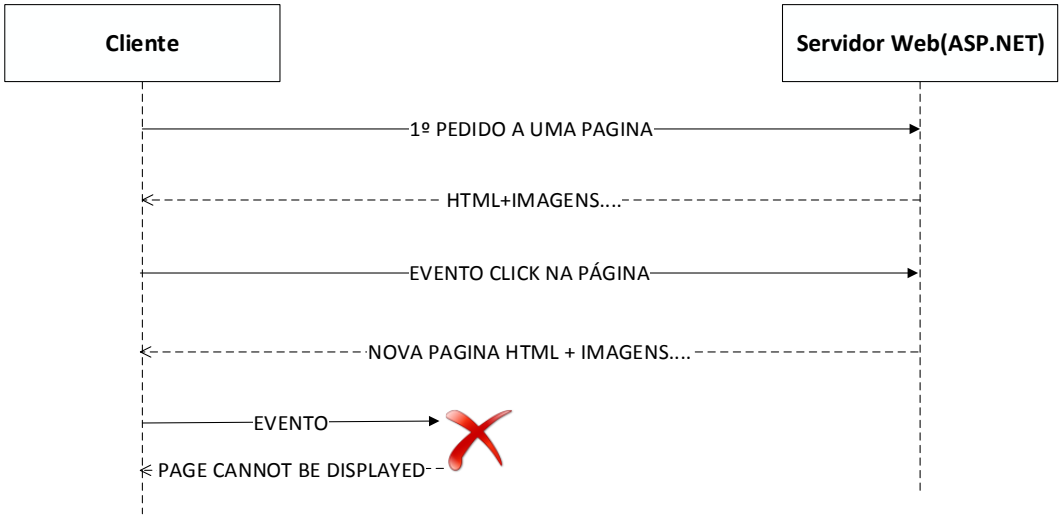


Figura 3.2– Cenário *Offline* na arquitetura *Server Side Web Applications*

A figura evidencia que este modelo é desadequando para cenários de fraca conectividade. No caso de ausência de conectividade (exemplificado com a cruz vermelha) o cliente (*Browser*) não conseguirá descarregar a página sendo apresentado um erro ao utilizador. Este erro deve-se ao facto das páginas serem criadas dinamicamente no servidor.

A figura 3.3 repete o mesmo cenário utilizando *AJAX*. Ao contrário do exemplo anterior que por cada pedido ao servidor é enviada uma nova página, esta técnica realiza pedidos *HTTP* de forma assíncrona permitindo que o *Browser* continue disponível às interações do utilizador.

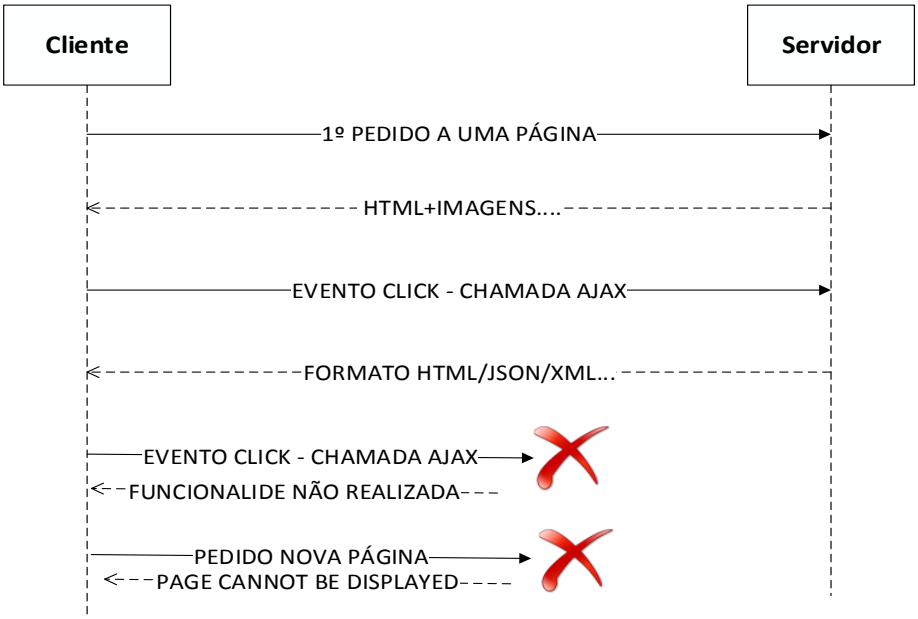


Figura 3.3 - Cenário *Offline* com *Ajax*

Como podemos observar pela figura num cenário de inexistência de conectividade o pedido *AJAX* não poderá ser realizado ao servidor. Como não existe persistência de dados locais a funcionalidade não é realizada.

Em síntese, ambos os modelos (*Server Side Aplications* e *Ajax*) são desadequados pois centralizam a componente dinâmica (lógica) da aplicação no lado do servidor e não possuem infraestrutura para persistência de dados locais.

O *Single Page Applications* (12) é um paradigma recentemente proposto para o desenvolvimento de aplicações *Web-Offline*. Baseia-se na existência de uma única página em que todas as atualizações são realizadas através de código *Javascript.* No primeiro pedido ao servidor é gravado localmente todo o código *HTML*, *Javascript* e *CSS* da página. Num cenário *Offline* estes recursos estão sempre disponíveis.

O modelo *SPA* pode concretizar-se acedendo a dados residentes no servidor ou dados locais (e.g., *Local Storage*) permitindo o funcionamento da aplicação em modo *Offline.* A figura 3.4 exemplifica esta abordagem.

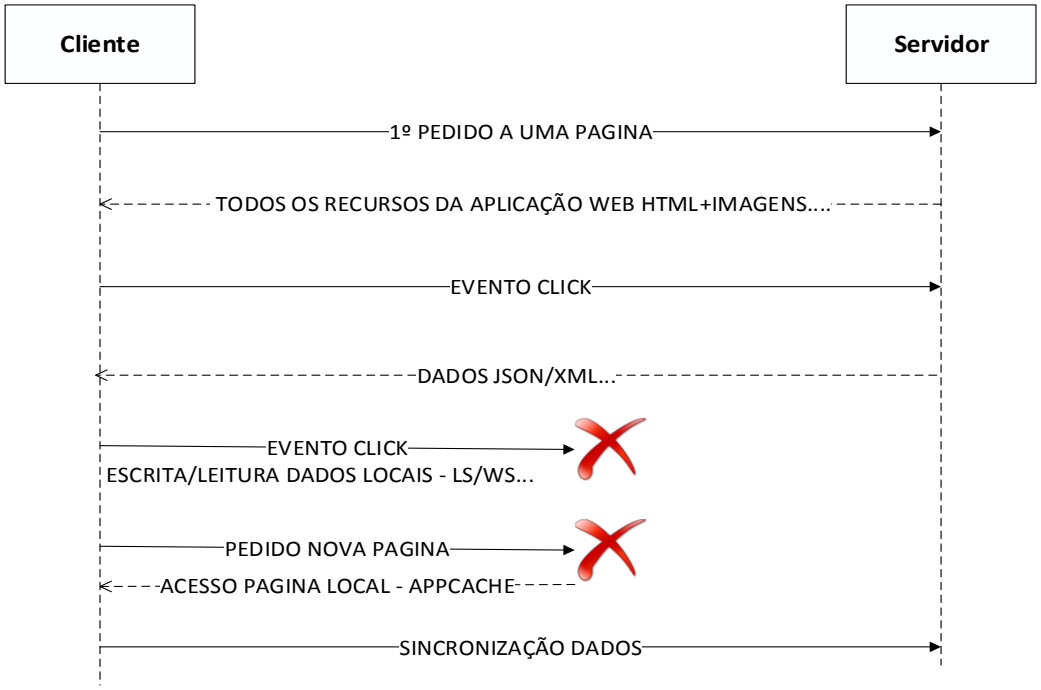


Figura 3.4- Cenário *Offline* em arquitetura *SPA*

Através da figura podemos concluir que mesmo num cenário *Offline* a aplicação continuará a funcionar. A lógica aplicacional está contida no lado do cliente e os recursos (código *HTML, Javascript,* CSS) foram serializados no primeiro pedido ao servidor.

Na inexistência de conectividade a página vai ser atualizada através de código *Javascript* com a criação de elementos *HTML DOM* associando estes elementos a dados locais. Os dados gerados em modo desconectado serão gravados localmente e posteriormente sincronizados assim que exista conectividade.

A consequência da implementação deste modelo é uma boa experiência de utilização da aplicação uma vez que ao contrário dos modelos anteriores a comunicação com o servidor faz-se apenas através de dados (exceto no primeiro pedido), diminuindo drasticamente a latência na comunicação cliente-servidor.

Por outro lado, o facto de a página ser carregada apenas uma única vez (no primeiro pedido) impede a libertação de memória já não utilizada (*Memory Leaks*), provocando um gradual aumento da memória alocada. A resolução deste problema passa por análise das variáveis em memória durante o fluxo do programa (por exemplo

utilizando o *Dev Tools* do *Browser Chrome*).Após essa análise terá que ser implementado código que permita a libertação das variáveis não utilizadas de forma explícita.

3.2. Modelo de Mercado

O processo de atribuição de tarefas para os diferentes utilizadores segue o modelo de mercado adotando o conceito de leilão.

O conceito de leilão pode definir-se como um mecanismo de negociação cujo principal objetivo é o de promover uma dinâmica transacional capaz de conduzir à revelação do preço de um determinado bem. Os quatro leilões considerados clássicos (13) são os seguintes:

- Leilão Fechado de primeiro preço (*First Price*)
- Leilão Fechado de segundo preço (Vickrey)
- Leilão Ascendente (*English*)
- Leilão Descendente (*Dutch*)

No leilão fechado cada participante lícita uma única vez em modo de "envelope fechado". Assim as diferentes propostas mantêm-se privadas até ao final do leilão; nunca exista informação pública.

No leilão fechado de primeiro preço o participante que licitar o valor mais alto ganha o leilão. No caso do leilão fechado de segundo preço o processo é exatamente o mesmo com a diferença do preço final de venda ser igual ao valor da segunda melhor oferta.

Em leilões ascendentes o preço é público e aumenta sucessivamente até que apenas um interessado continue ativo. Os leilões descendentes funcionam de maneira inversa: o processo inicia-se a um preço que vai decrescendo até que o primeiro participante manifeste sua intenção de realizar a transação.

Num leilão aberto (tanto ascendente como descendente) podem existir várias fases de licitação. Assim, o utilizador que possuir melhores condições de conectividade fica numa situação privilegiada (face aos restantes utilizadores) para vencer o leilão pois

poderá licitar um maior número de vezes no leilão. A tabela 2.1 apresenta esta relação.

	Intervenção no leilão	Vantagem negocial
Boa conectividade	Alto	Alto
Fraca conectividade	Baixo	Baixo

Tabela 3.1– Tabela comparativa de utilizadores com conectividades diferentes em leilão

No leilão fechado como existe apenas uma licitação, e considerando uma "janela" de tempo aceitável para a duração do leilão todos os utilizadores terão mais possibilidades de participar no leilão.

Num cenário de fraca conectividade os leilões fechados são mais adequados por aumentarem a equidade na participação quando comparados com o processo de participação num leilão aberto.

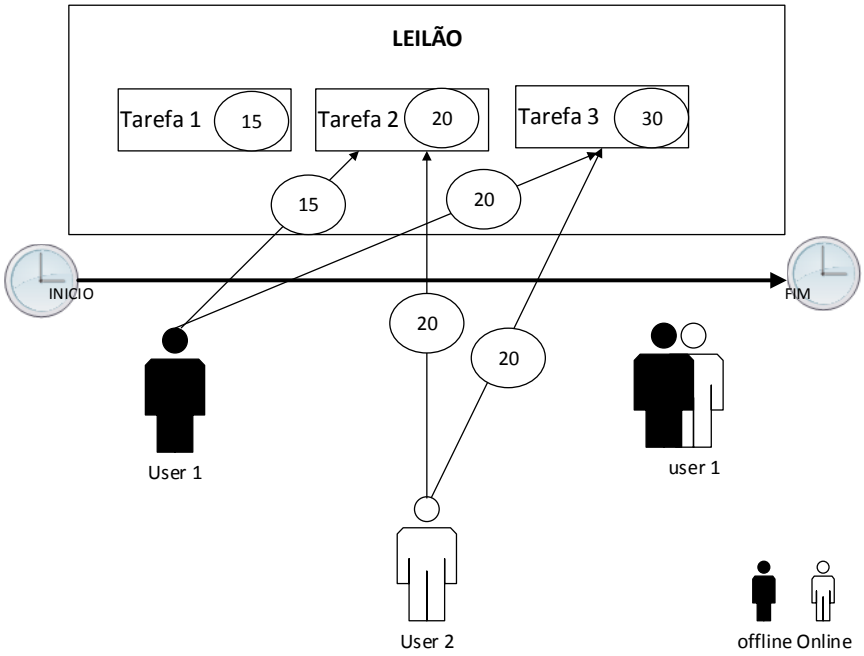
Assim o modelo de mercado proposto baseia-se no modelo de leilão fechado em que existe uma única licitação (a considerar para cada utilizador). Foram implementados dois tipos de leilão: o *Vickrey* e o *First Price*.

Em geral, num mecanismo de leilão o participante que licitar o maior valor será o vencedor. No contexto deste projeto o processo é simétrico. Quando o utilizador lícita um valor para realizar uma tarefa, está a indicar o preço que pretende receber para a sua realização. O licitador por seu lado pretende pagar o mínimo valor para a realização da tarefa.

Assim no leilão *First Price* o utilizador que licitar o menor valor será o vencedor do leilão. No caso do leilão tipo *Vickrey* a tarefa será igualmente atribuída à licitação mais baixa com a diferença que o valor de licitação considerado para o vencedor será o segundo menor valor.

Os leilões implementados suportam a capacidade *Offline*. Neste cenário um utilizador que realize uma licitação sem conectividade mantém persistência local. Caso o utilizador consiga restabelecer a conectividade antes do leilão terminar, a licitação será considerada para a atribuição da tarefa.

A figura 3.5 apresenta um cenário ilustrativo do processo de leilão para a afetação das tarefas aos utilizadores.



Resultado do Leilão Vickrey/FirstPrice
Tarefa 1: Não alocada
Tarefa 2: User 1 – Valor tarefa: 15/20
Tarefa 3: User 1 – Valor tarefa: 20/20

Figura 3.5– Funcionamento do leilão adotado

Na figura está representado um leilão com três tarefas e dois utilizadores (*User 1* e *User 2*]. O utilizador preto simboliza um utilizador sem conectividade. O utilizador branco com conectividade.

Cada tarefa possui um teto máximo (valor de referência da tarefa) de licitação que corresponde a 15 para tarefa 1, 20 tarefa 2 e 30 tarefa 3. A linha entre os dois relógios corresponde ao tempo do leilão.

O *User 1* foi o primeiro utilizador a licitar, licitando o valor 15 para tarefa 1 e 20 para tarefa 3. No momento de licitação o utilizador encontrava-se numa situação *Offline* ficando as suas licitações gravadas localmente.

O *User 2* licitou o valor 20 para a tarefa 2 e 20 para a tarefa 3. Este utilizador encontrava-se com conectividade e as suas licitações foram persistidas no servidor.

Antes de terminar o leilão o *User 1* recuperou conectividade sincronizando as suas licitações com o servidor. No final do leilão as tarefas 2 e 3 ficaram alocadas ao *User 1* e a tarefa 1 não ficou alocada a nenhum utilizador.

A tarefa 1 não ficou alocada uma vez que não recebeu qualquer licitação, estando disponível para licitação no próximo leilão. A tarefa 2 ficou alocada ao *User 1* pois este licitou menor valor que o *User 2*.

A tarefa 3 ficou alocada ao *User 1*, pois apesar de licitar o mesmo valor que o *User 2*, a sua licitação foi efetuada primeiro. As licitações do *User 2* foram enviadas para o servidor antes das licitações do *User 1* (uma vez que este se encontrava sem conectividade), contudo é sempre registado a data da licitação mesmo numa situação sem conectividade.

A figura 3.5 apresenta os valores licitados considerados para os vencedores. Para o leilão *First Price* o valor de licitação é o menor valor licitado, o que corresponde ao valor 15 para a tarefa 2 e 20 para a tarefa 3. No caso do leilão *Vickerey* o valor de licitação é o segundo valor mais baixo, que é 20 para a tarefa 2 e 20 para tarefa 3.

3.2.1. Parametrização do Leilão

Na criação de um leilão deverá ser indicado o tipo de leilão bem como a sua data de início (D_{inicio}) e fim (D_{final}). Cada leilão tem um parâmetro (*delta*) que define um intervalo de tempo dentro de um leilão, criado com dois propósitos:

1. Impedir que sejam adicionadas tarefas ao leilão quando $D_{final}-D_{inicio} <= D_{elta}$
2. Atribuir penalização ao utilizador que rejeitar tarefas (atribuídas em leilão anterior) quando $D_{final}-D_{inicio} <= D_{elta}$ (no leilão corrente).

O parâmetro “delta” foi criado de forma a não penalizar os utilizadores em situação de fraca conectividade. Na primeira situação o objetivo é evitar a entrada de novas tarefas numa janela temporal curta para licitação, prejudicando os utilizadores com fraca conectividade. Na segunda situação pretende-se não penalizar os utilizadores que por alguma razão tencionam rejeitar alguma tarefa atribuída mas por razões de fraca conectividade não o conseguem fazer nesse momento.A figura 3.6 ilustra um exemplo para a primeira situação descrita.

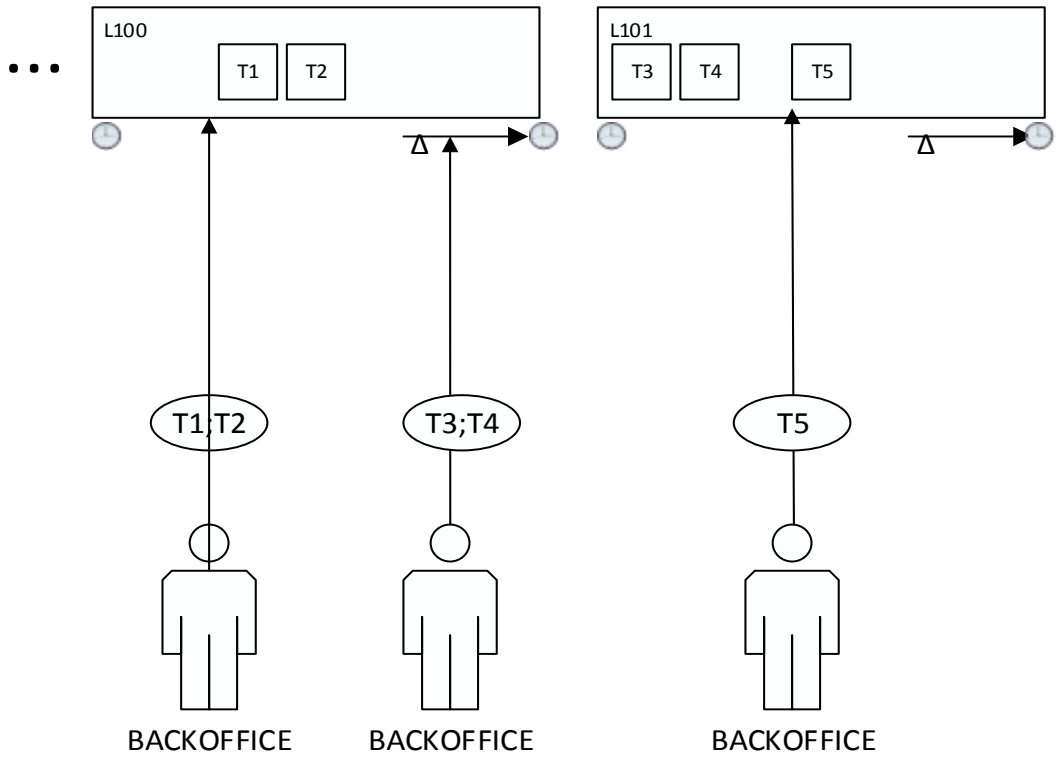


Figura 3.6– Parâmetro delta – inclusão de novas tarefas

A inclusão de novas tarefas aos utilizadores é realizada em *Backoffice*. Através da figura podemos verificar que as tarefas *T1* e *T2* são disponibilizadas no leilão 100 uma vez que foram enviadas antes do intervalo *delta* ser atingido. Por seu lado as tarefas *T3* e *T4* não foram incluídas no leilão 100 ficando apenas disponíveis no leilão seguinte (101), uma vez que o intervalo Delta foi alcançado.A figura 3.7 ilustra a segunda situação descrita.

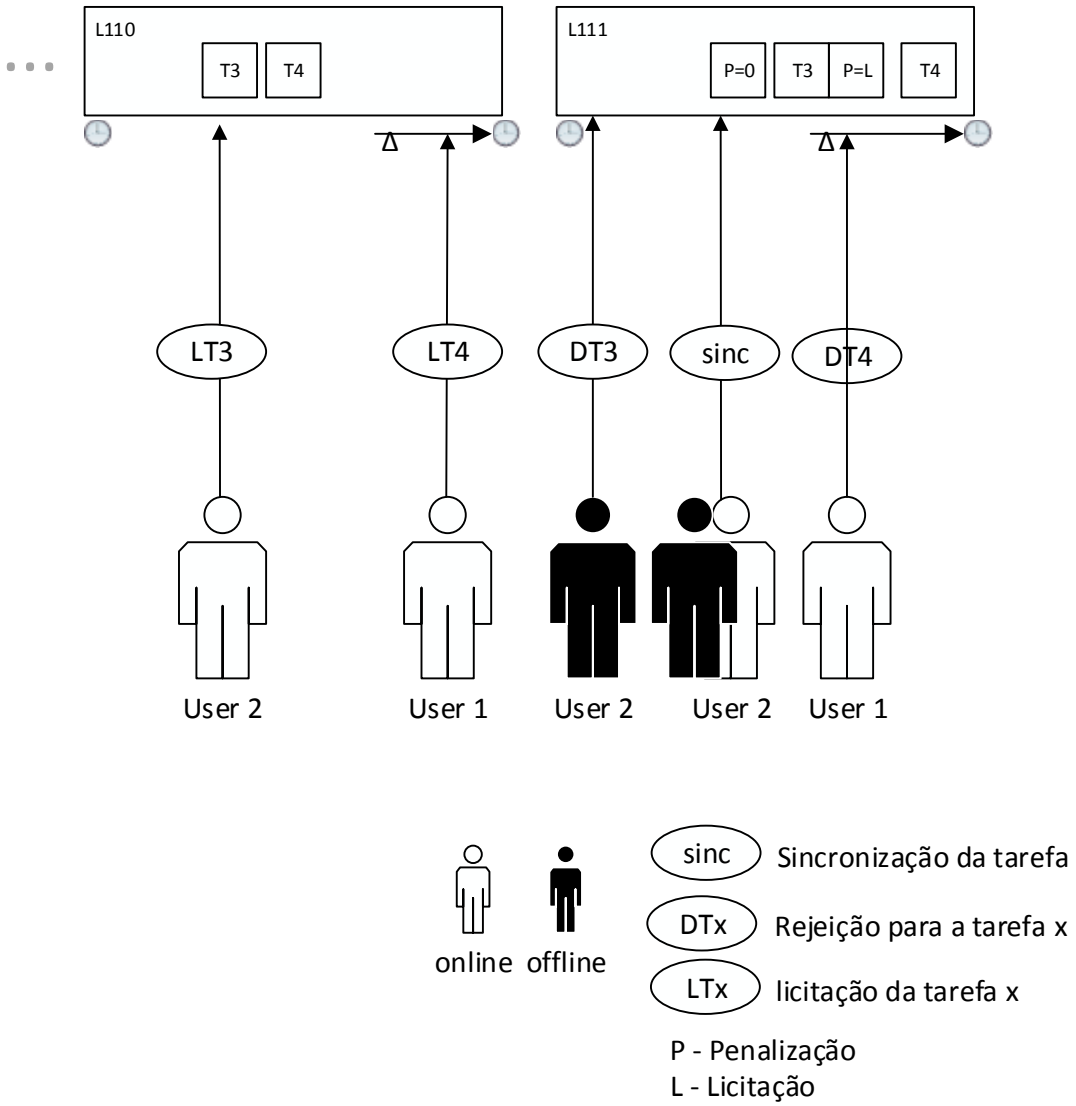


Figura 3.7– Parâmetro delta – rejeição de tarefas atribuídas

No leilão 111 os utilizadores *User 1* e *User 2* rejeitaram as tarefas atribuídas no leilão anterior (110). O *User 2,* inicialmente sem conectividade regista localmente essa operação. Quando é restabelecida a ligação para o *User 2* é sincronizado com o servidor. Dado que a sincronização é realizada antes de atingir o intervalo *Delta* o *User 2* não será penalizado. Por seu lado, o utilizador *User 1* é penalizado por rejeitar a tarefa dentro do intervalo delimitado por *delta*. O valor da penalização é igual ao valor da licitação.

3.3. Construção de uma Framework

Com o objetivo de uma modelação mais refinada do domínio do problema de forma a conduzir o programador a uma escrita de código organizada e sistemática no lado do cliente procedeu-se a implementação de uma *Framework*. Esta seção detalha cada componente da *Framework* descrevendo as suas principais funções.

3.3.1. Máquina de Estados

A máquina de estados coordena o fluxo da aplicação de acordo com os seus estados internos. O comportamento funcional do sistema num dado momento é realizado segundo o estado corrente da máquina de estados. Sempre que existe uma transição de estado (desencadeado por um evento) é atualizado o objeto estado corrente. A figura 3.8 representa o diagrama de transições de estados para alguns eventos.

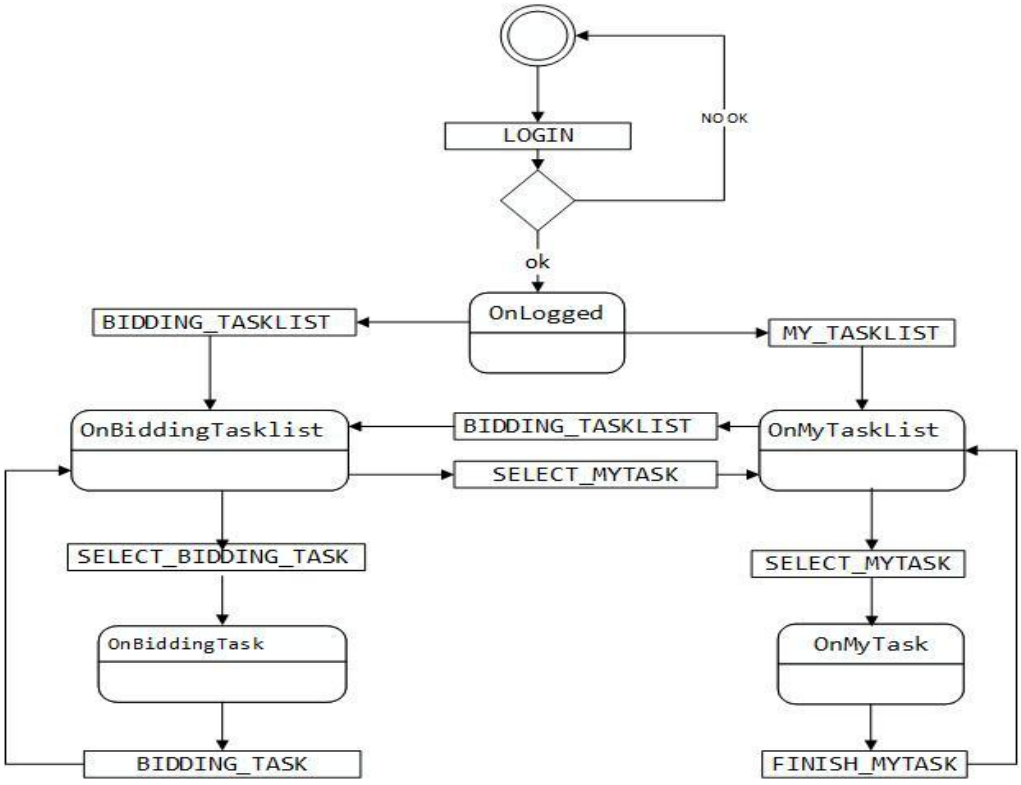


Figura 3.8– Máquina de estados - eventos estados e transições

Os retângulos representam eventos, as caixas estados e as setas as transições estado - evento - novo estado. Através da figura observa-se que todo o fluxo da aplicação é coordenado na máquina de estados oferecendo uma maior robustez à aplicação. Se por algum erro da interface gráfica for despoletado um evento inadvertido, a máquina

de estados verifica e valida se o evento deverá ser executado segundo o estado corrente, evitando a existência de erros.

Os eventos, estados e as transições de estado são definidos no ficheiro de configuração da máquina de estados. A figura 3.9 demostra a sua parametrização.

```
var EVENT_SET = {

    BIDDING_TASKLIST: { value: 0, name: "biddingTaskList" },
    SELECT_BIDDING_TASK: { value: 16, name: "selectBiddingTask" },
    BIDDING_TASK: { value: 1, name: "biddingTask" }
        …}
var STATE_SET = {

    LOGGED: "OnLogged",
    BIDDING_TASKLIST: "OnBiddingTasklist",
    BIDDINGTASK: "OnBiddingTask",
    …}

var TRANSITION_FUNCTION = [
{ currState: STATE_SET.MYTASK,
allowEvents: [
    { eventName: EVENT_SET.ANSWER_ACTIVITY.name, nextState: STATE_SET.MYTASK },
    { eventName: EVENT_SET.BACK.name, nextState: STATE_SET.MYTASKLIST },
    { eventName: EVENT_SET.FINISH_MYTASK.name, nextState: STATE_SET.MYTASKLIST}] }
    …]
}
```

Figura 3.9– Ficheiro de configuração da máquina de estados

Para configurar a máquina de estados é necessário indicar três parâmetros:

- *EVENT_SET*: todos os eventos possíveis de serem gerados na aplicação
- STATE_SET: todos os estados possíveis da aplicação
- TRANSACTION_FUNCTION: dado o estado corrente são identificados os eventos possíveis de serem executados e o respetivo novo estado da máquina de estados

Na sua criação, a máquina de estados carrega para memória esta parametrização coordenando a aplicação com base nesta informação.

3.3.2. Contentor de Modelos

O contentor de modelos é o repositório de objetos que representam os dados. É constituído por dados persistentes provenientes do servidor ou persistidos localmente.

O contentor de modelos disponibiliza uma interface que permite o registo aos objetos que representam dados. No processo de registo cada objeto indica uma chave, que constituirá a sua identificação no contentor de modelos.

Cada estado acede a dados sempre através do contentor de modelos. A figura 3.10 demostra o modo de funcionamento do contentor de modelos.

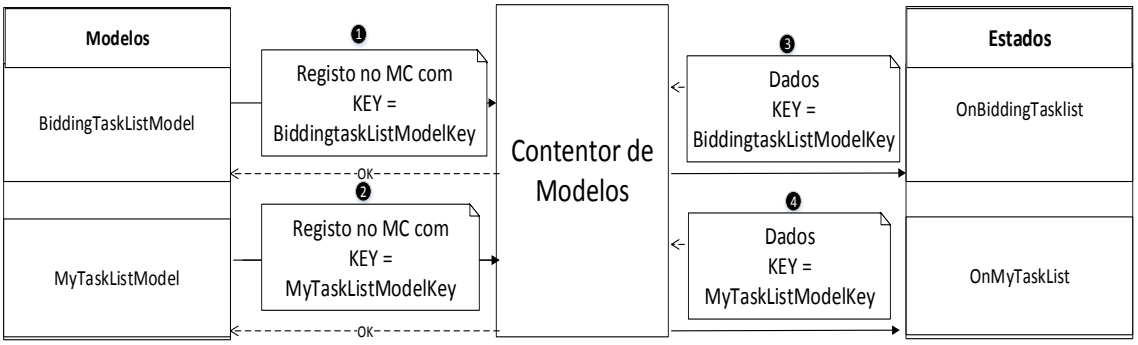


Figura 3.10– Contentor de modelos – acesso a dados persistentes

Como ilustrado na figura, o processo inicia-se com o registo dos objetos "Modelo" no contentor. Após armazenados, o acesso a dados é sempre realizado através do contentor. Esta solução permite um desacoplamento na relação estado-dados permitindo que um estado aceda a dados sem necessitar de conhecer o objeto que o representa.

Além de objetos persistentes, o contentor de modelos permite alocar dados voláteis relativos ao contexto de utilização da aplicação (e.g., dados relativos ao preenchimento de um formulário). Tal como os persistentes ("Modelo"), os objetos voláteis ("Contexto") são identificados através de uma chave. A figura 3.11 demostra um exemplo da utilização deste tipo objetos.

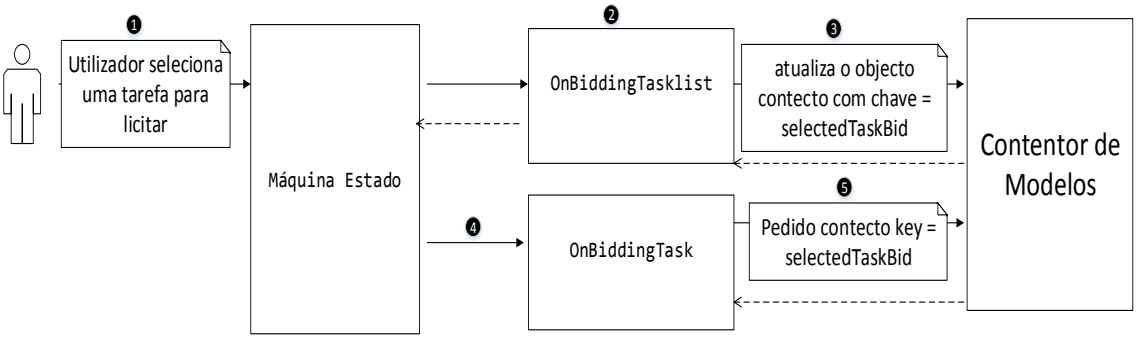


Figura 3.11– Contentor de modelos – acesso a dados de "contexto"

O processo é iniciado com o envio de um evento (seleção de uma tarefa para licitação) para a máquina de estados. O estado corrente, *OnBiddingTaskList*, atualiza o objeto "Contexto" com tarefa selecionada no contentor de modelos. O evento enviado despoleta uma transição da máquina de estados para um novo estado, *OnBiddingTask*.

Este novo estado obtém a tarefa selecionada pelo utilizador através do objeto "Contexto" presente no contentor de modelos.

Os objetos que representam dados na *Framework* ("Modelo" e "Contexto") implementam este padrão *MVVM*. Assim, cada alteração na vista é automaticamente propagada para o "Modelo" de dados associados. No sentido inverso o comportamento é idêntico, a alteração dos dados é automaticamente atualizada na interface gráfica (*Duplex Binding*). A figura 3.12 ilustra um exemplo deste comportamento

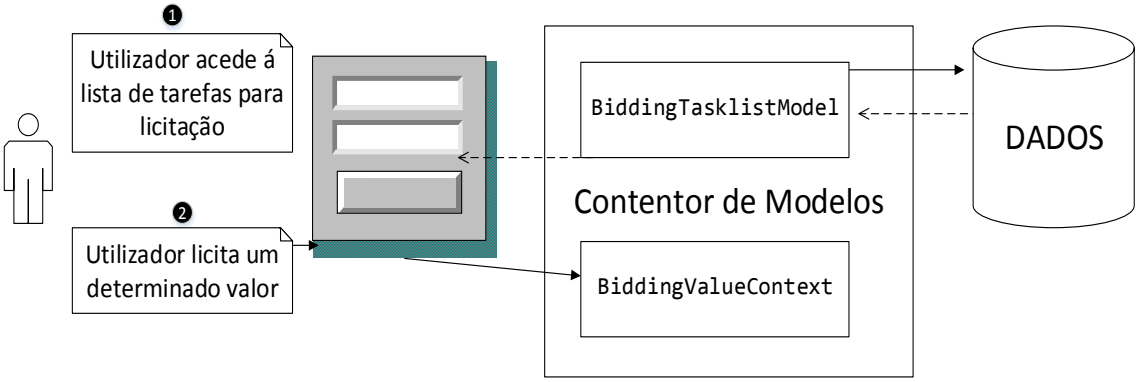


Figura 3.12 – Paradigma *MVVM* no contentor de modelos

A lista de tarefas disponíveis ao utilizador é obtida atualizando os dados do objeto "Modelo" (*BiddingTasklistModel*), diretamente no servidor se existir conectividade ou localmente na inexistência de conectividade. Dado que o objeto implementa o padrão *MVVM,* a sua atualização é automaticamente propagada ao controlo gráfico associado existente na interface gráfica do utilizador.

No sentido contrário, quando o utilizador lícita um valor para a tarefa é imediatamente passado ao objeto contexto que o representa, *BiddingValueContext*.

3.4. Motor de Criação de Tarefas

O motor de criação de tarefas pretende generalizar o conceito de tarefa a diferentes problemas. Neste projeto o conceito de tarefa é aplicado a dois contextos

- Tarefas do tipo auditoria - conjunto de pressupostos que deverão ser verificados de forma a aferir se o técnico realiza o seu trabalho corretamente.

- Tarefas com conteúdo pedagógico - conjunto de questões sobre um determinado conteúdo programático (e.g., Matemática, Português)

Uma tarefa é caracterizada pelo conjunto de atividades e a figura 3.13 representa o modelo de uma tarefa

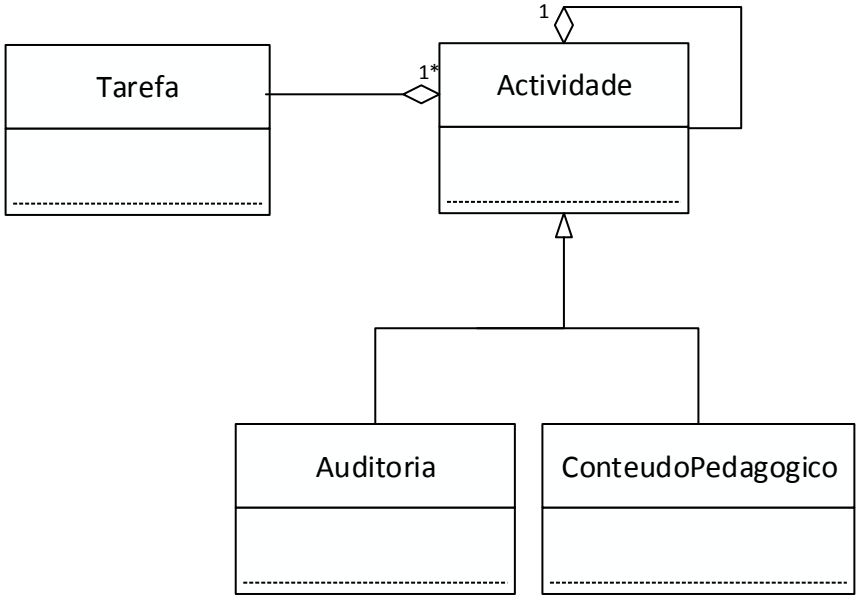


Figura 3.13– Modelo da tarefa

Observando a figura 3.13 constata-se que uma atividade pode ser especializada em diferentes contextos (e.g., Auditorias, Conteúdo Pedagógico). As atividades relacionam-se segundo resposta efetuada (execução), podendo diferentes respostas dar origem a diferentes atividades.

Por exemplo no contexto das auditorias, se numa atividade o utilizador verificar que o técnico não possuia equipamento de calibragem, não fará sentido a realização de outra atividade para medição desse equipamento. Na posse do equipamento, a atividade de medição deveria ser realizada.

A relação entre atividades é realizada pela pessoa que cria a tarefa (em *BackOffice*), configurando-a da forma mais apropriada. A figura 3.14 ilustra um exemplo de relações entre atividades em forma de árvore.

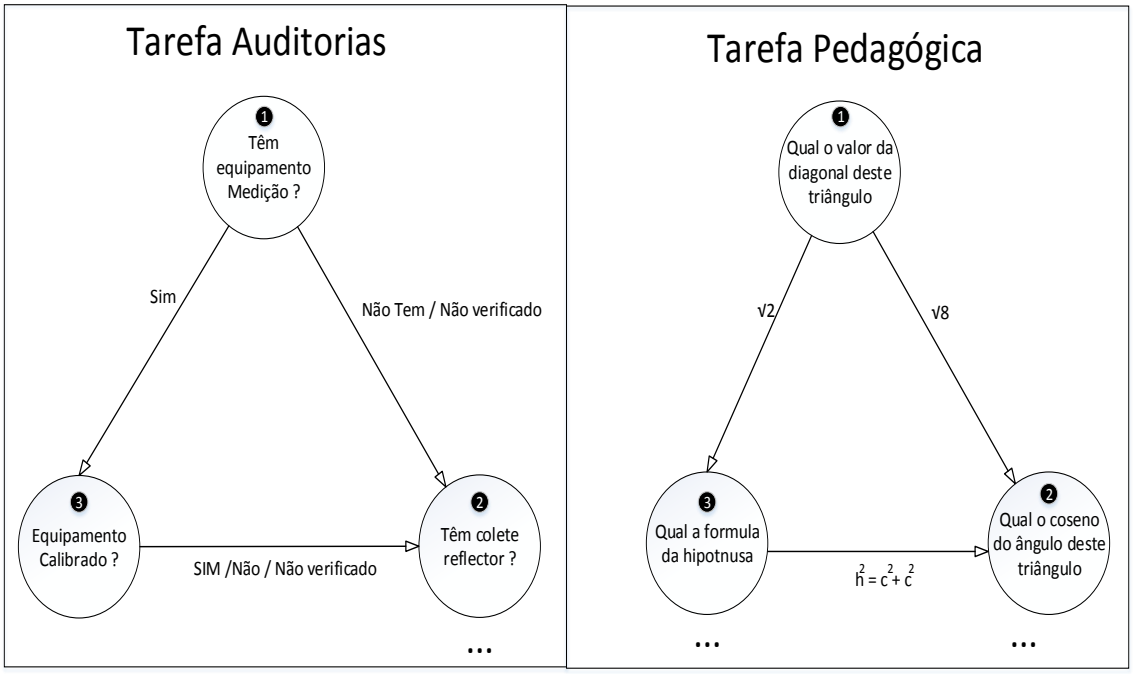


Figura 3.14 – Árvore de tarefas para dois domínios diferentes do problema

A figura exemplifica duas tarefas criadas para domínios de problemas diferentes. As atividades são representadas através dos nós e as respostas pelos arcos.

Através da figura percebe-se que a próxima atividade a realizar está relacionada com a execução da atividade corrente. Na tarefa de auditoria, se para a atividade 1 a resposta for "Sim" a próxima atividade será a 3, caso contrário a 2.

Na tarefa de conteúdo pedagógico, se na atividade 1 a resposta não for correta (√8), o utilizador vai ser conduzido a outra atividade com o objetivo de readquirir os conhecimentos que não demostrou na atividade anterior.

3.4.1. Atividades – Tipos de Resposta

As atividades podem ser realizadas segundo dois tipos de resposta. O primeiro é de resposta fixa. Na criação da tarefa são configuradas as hipóteses possíveis de resposta para essa atividade. Na figura 3.14 para tarefa de auditoria na atividade 1 (tem equipamento de medição), o tipo de resposta por parte do utilizador será o de resposta fixa. Neste caso têm apenas disponível as hipóteses: "Sim", "Não Têm", "Não verificado".

O segundo tipo de resposta é editável. É apresentada uma caixa de texto ao utilizador para indicar um valor. Na figura 3.14 para tarefa de auditoria na atividade 2

(Equipamento Calibrado) o tipo de resposta do utilizador é editável, com a indicação do valor de calibragem.

3.4.2. Avaliação da Tarefa

A realização da tarefa por parte do utilizador é alvo de uma avaliação. Essa avaliação é ponderada através de dois vetores, eficácia e eficiência. A eficácia máxima é conseguida com a realização de todas as atividades corretamente. A eficiência será maior quanto mais rápido realizar tarefa.

Na criação de uma atividade é indicado o tempo esperado para a sua a realização e os pontos possíveis de serem obtidos pelo utilizador. O valor dos pontos está relacionado com a execução da atividade.

Para o caso das auditorias se o utilizador numa atividade não verificar um pressuposto (opção Não verificado) receberá uma pontuação menor.

No exemplo das tarefas de conteúdo pedagógico, a pontuação máxima será obtida se a resposta à atividade estiver correta. As tabelas 3.2 e 3.3 apresentam a parametrização de tarefas diferentes para estes dois contextos.

AT	Fixa/Editável	P	Prox AT
1	Sim	10	3
1	Não Tem	10	2
1	Não Verificado	0	2
2	…	…	…
3	x >=10 && x<=20	0	2
3	x<10	0	2
3	x>20	0	2
…	…	…	…
Max		40	

Tabela 3.2 - Tabela representativa da configuração de uma tarefa de auditoria

AT	Fixa/Editável	P	Prox AT
1	√8	20	3
1	√2	0	2
2	..	..	..
3	=60	20	2
3	!=60	0	2
…	…	…	…
Max		40	

Tabela 3.3 - Tabela representativa da configuração de uma tarefa pedagógica

A tabela 3.2 representa a parametrização de uma tarefa no contexto das auditorias, e a tabela 3.3 no conteúdo pedagógico. Uma atividade (coluna *AT*) tem várias hipóteses de execução, que corresponde a cada linha da tabela. Por cada hipótese é indicado o valor dos pontos obtidos (coluna *P*) e a próxima atividade a ser realizada (coluna *Prox AT*).

Cada atividade pode ser efetuada por resposta fixa ou editável (coluna Fixa/Editável). No caso de resposta fixa, é contabilizado o valor da pontuação para a hipótese escolhida. Para a resposta editável são avaliadas as várias expressões lógicas existentes para essa atividade. Na avaliação é utilizado o valor editado do utilizador como parâmetro na expressão. A expressão que devolver um valor verdadeiro será a considerada.

Para que apenas exista uma única expressão lógica verdadeira, duas condições terão que se verificar. Em primeiro lugar o domínio de valores de cada expressão não se pode interceptar. O não cumprimento deste requisito pode provocar a existência de mais que uma expressão lógica verdadeira e consequentemente a indefinição da resposta a considerar. Em segundo lugar a reunião de todas as expressões terá que devolver um valor verdadeiro.

Na criação da tarefa é contabilizado o valor máximo que um utilizador pode obter. Esse valor corresponde ao somatório de todas as atividades realizadas corretamente (linha *max*, coluna *P* das tabelas 3.2 e 3.3). Esse valor corresponde ao valor de referência e ao teto máximo de licitação para a tarefa.

A eficácia da tarefa de um utilizador é calculada através da relação pontos obtidos e pontuação máxima, relativamente a licitação realizada. A fórmula em baixo corresponde à avaliação da eficiência de um utilizador numa tarefa.

$$Eficiencia = \frac{1-((T_{eto} - P_{obtidos}) * L_{realizada})}{T_{eto}}$$

O tempo esperado para a realização da tarefa é calculado somando todos os tempos configurados nas atividades. A eficiência é calculada na relação tempo despendido e tempo esperado, relativamente à licitação realizada. A fórmula em baixo corresponde à avaliação da eficácia de um utilizador numa tarefa.

$$Eficacia = \frac{(T_{esperado} - T_{realizado})}{T_{esperado}} * L_{realizada}$$

O valor mínimo de eficiência para um utilizador é zero, ou seja, se o utilizador despender mais tempo na tarefa que o tempo máximo não será penalizado. A avaliação do utilizador corresponde à soma da eficiência com a eficácia como demostra a fórmula em baixo.

$$Pontuação = Eficiencia + Eficacia$$

3.5. Modelo Servidor

A componente servidora utilizada neste projeto foi o *NodeJS*. O *NodeJS* é uma plataforma construída sobre *V8 Javascript Engine* que permite a utilização da linguagem *Javascript* no lado servidor.

Acesso	Ciclos de processador
Cache L1	3
Cache L2	14
RAM	250
Disco	41000000
Rede	240000000

Tabela 3.4 - Dados fornecidos na apresentação do Node.js (2008-11-08)

Analisando a tabela 3.4 constatasse que o custo do processador para acesso a disco ou rede é exponencialmente superior ao acesso à memória (*cache* e *RAM*). Tirando

partido da programação assíncrona da linguagem *Javascript*, a plataforma *Nodejs* permite que todo o acesso *I/O* não seja realizado de forma bloqueante, utilizando um modelo orientado a eventos para esse efeito. A figura 3.15 ilustra a utilização de invocações assíncronas em *Javascript*

```
function teste(){ console.log('teste'); }
1 - setInterval(teste,0);
2 -console.log('antes de teste');
Output:
antes de teste
teste
```

Figura 3.15- Programação assíncrona em *Javascript*

O troço de código possui duas instruções identificadas pelo número 1 e número 2. Analisando o *output* observa-se que apesar da chamada à função *teste* (instrução 1) ser anterior à instrução 2 a sua execução é posterior. Esta situação acontece uma vez que a chamada à função *teste* é realizada de forma assíncrona. O *NodeJS* tira partido deste modelo de programação utilizando chamadas assíncronas para acesso *I/O*.
O *NodeJS* segue um paradigma de programação orientada a eventos. Neste paradigma existe três participantes

- *Event Producer*: participante que produz o evento
- *Event Consumer:* executor do evento
- *Event Listener* : observador do evento

Um evento em *NodeJS* é um acesso *I/O*. Por cada acesso *I/O* é criado um evento (*Event Producer*) que é colocado na fila de eventos (*Event Queue*) com o respetivo nome e *Callback*. O *Callback* é o troço de código realizado pelo observador (Event *Listener*) após a realização do evento.

A execução do evento é realizada por uma *Thread* existente no *Thread Pool.* Após realizado o evento é colocado na fila de eventos com o resultado da sua execução.

A *Event Loop* é o componente responsável pela gestão da fila de eventos. Utilizando um único fio de execução (*Thread*), iterativamente consulta a fila de eventos de modo a aferir se existem eventos por executar ou eventos já executados. Os eventos por executar são enviados aos executores de forma a serem realizados. Os eventos

executados são enviados aos observadores para que executem o seu *callback*. A figura 3.16 demostra esta arquitetura.

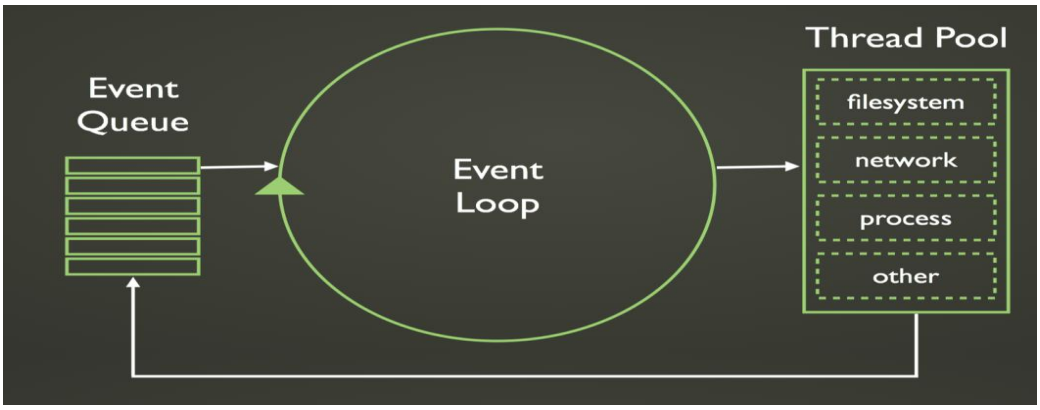


Figura 3.16– Arquitetura *NodeJS* (apresentação do Node.js em 2008-11-08)

O *NodeJS* é a prova que é possível ter uma arquitetura baseada em eventos no lado do servidor. Esta arquitetura permite alcançar níveis de desempenho consideráveis.

3.6. Verificação da Conectividade – Online/Offline

O protocolo *HTTP* é um protocolo *Request-Response* no qual o cliente envia um pedido ao servidor que o processa e retorna uma resposta. Neste modelo a comunicação é sempre iniciada pelo cliente e nunca pelo servidor.

Uma das funcionalidades oferecidas pela aplicação móvel é a indicação ao utilizador se num dado momento existe conectividade com o servidor. Para a implementação desta funcionalidade utilizando o protocolo *HTTP* existem três técnicas possíveis de implementação

- *Polling*
- *LongPolling*
- *Push*

O *Polling* é uma técnica na qual o cliente está constantemente a realizar pedidos ao servidor. Dependentemente da periocidade dos pedidos, esta técnica pode ser uma boa ou má solução. Para a funcionalidade implementada o intervalo entre pedidos pode ser elevado. Num cenário com informação mais volátil (e.g., informação sobre cotações em bolsa) o intervalo de pedidos teria que ser encurtado. O problema desta

técnica é o facto de muitos pedidos se tornarem inúteis se não existirem atualizações no servidor, aumentando o consumo de banda.

O *Long Polling* é uma técnica na qual se utiliza *Polling* mas numa forma mais eficiente. Na realização de um pedido ao servidor a resposta não é imediata. Existe um tempo de espera no servidor antes de enviar a resposta. Durante esse tempo poderão existir atualizações no servidor que serão incluídas na resposta ao cliente. A vantagem desta técnica em relação ao *Polling* é a diminuição de pedidos ao servidor. A desvantagem é que durante o período de espera poderão existir atualizações e o cliente apenas as recebe no final do tempo.

O *Push* é uma técnica mais sofisticada que as anteriores. Este modelo de comunicação inverte o modelo tradicional *HTTP* no qual o servidor apenas envia respostas a pedido do cliente. Esta técnica permite que o servidor envie dados ao cliente (*Push*) sem que este tivesse solicitado qualquer pedido. Este mecanismo é utilizado por duas tecnologias utilizadas nos dias de hoje - *Comet* e *Web Sockets*.

A tecnologia *Web Sockets* (14) define um canal de comunicação *Full-Duplex* sobre um único *Socket*. Isso significa que tanto o cliente como o servidor podem realizar pedidos ou respostas. A tecnologia *Comet* (15) funciona sobre o padrão *Publish/Subscrive* no qual o cliente regista-se em eventos do servidor e este notifica-o na sua ocorrência. Estas duas tecnologias utilizam sempre um canal aberto de comunicação entre o cliente e o servidor.

Para a implementação da funcionalidade a opção recaiu pelo *LongPolling*. A principal razão para esta escolha deve-se à facilidade de implementação. Utilizando apenas *Javascript*, é realizado um pedido assíncrono através da função *SetTimeout*, passando como argumento o tempo de espera. Outra justificação é o facto de não existir necessidade de verificar em tempo real se existe ou não conectividade com o servidor. Uma verificação mais demorada não cria qualquer impacto na funcionalidade.

A utilização do mecanismo *Push* na implementação desta funcionalidade implicaria a integração de mais tecnologia no projeto aumentado as dependências tecnológicas na aplicação.

3.7. Persistência

Com a massificação da *Web* surgiram aplicações com milhares de utilizadores que manipulam grandes volumes de dados (via escritas e leituras nas bases de dados). Os sistemas *NoSQl* (16) surgiram para responder a essa nova necessidade. A ideia destes sistemas é abdicar das restrições *ACID* implementadas nos *SGBDs* relacionais de forma a obter maior performance e escalabilidade.

Estes sistemas estão divididos em quatro grupos

- *Key-Value Stores:* armazenam valores e um índice para os encontrar, baseado numa chave
- *Document-Stores*: armazenam documentos que são indexados.
- *Column-Stores*: o armazenamento é orientado ao atributo (coluna)
- *Graph-DataBase*: os dados são armazenados em forma de grafo

O sistema adotado para persistência foi o *MongoDB* (17) que pertence à família de sistemas *NoSQL*. A tabela 3.5 relaciona os conceitos deste sistema com o modelo relacional.

Termos e conceitos Relacional	Termos e conceitos *MongoDB*
Tabela	Coleção
Linha	Documento
Coluna	Atributo
Índice	Índice
Join	-
Transação	-

Tabela 3.5- Tabela com a relação de termos no *MongoDB* com sistemas relacionais

Se repararmos não existe mapeamento para o *MongoDB* para os conceitos *Join* e *Transação*. A ausência dessas operações é motivada pelo impacto na redução do desempenho que a sua presença acarreta.

Ao nível transacional, o *MongoDB* apenas garante consistência ao nível do documento. A relação entre dados (*Join*) também só poderá existir no documento, e é realizada através de aninhamentos. O troço de código abaixo ilustra essa forma de relação.

```
task:{
    id:1,
    name: "Auditorias RCO2"
    Activities:[
        { id:1,value:"Tem Equipamneto de medição"}
        { id:2...}
    ]
}
```

Figura 3.17 – Representação de uma tarefa no *MongoDB*

O exemplo do troço acima representa a relação entre tarefas e atividades. Uma tarefa pode ter uma ou mais atividades. As atividades são aninhadas dentro do documento tarefa através de um *Array*.

Os dados no *MongoDB* são guardados num formato binário, denominado *BSON*. Este formato suporta além dos principais tipos primitivos tipos de dados de grande dimensão como imagens ou vídeo.

No *MongoDB* a modelação dos dados segue um esquema dinâmico ao contrário do modelo relacional onde o esquema é fixo oferecendo uma maior flexibilidade na implementação. A consistência de dados não é garantida de forma a maximizar o desempenho.

4. Implementação do Modelo

Neste capítulo são apresentados os detalhes de implementação. A figura 4.1 apresenta a arquitetura implementada neste projeto.

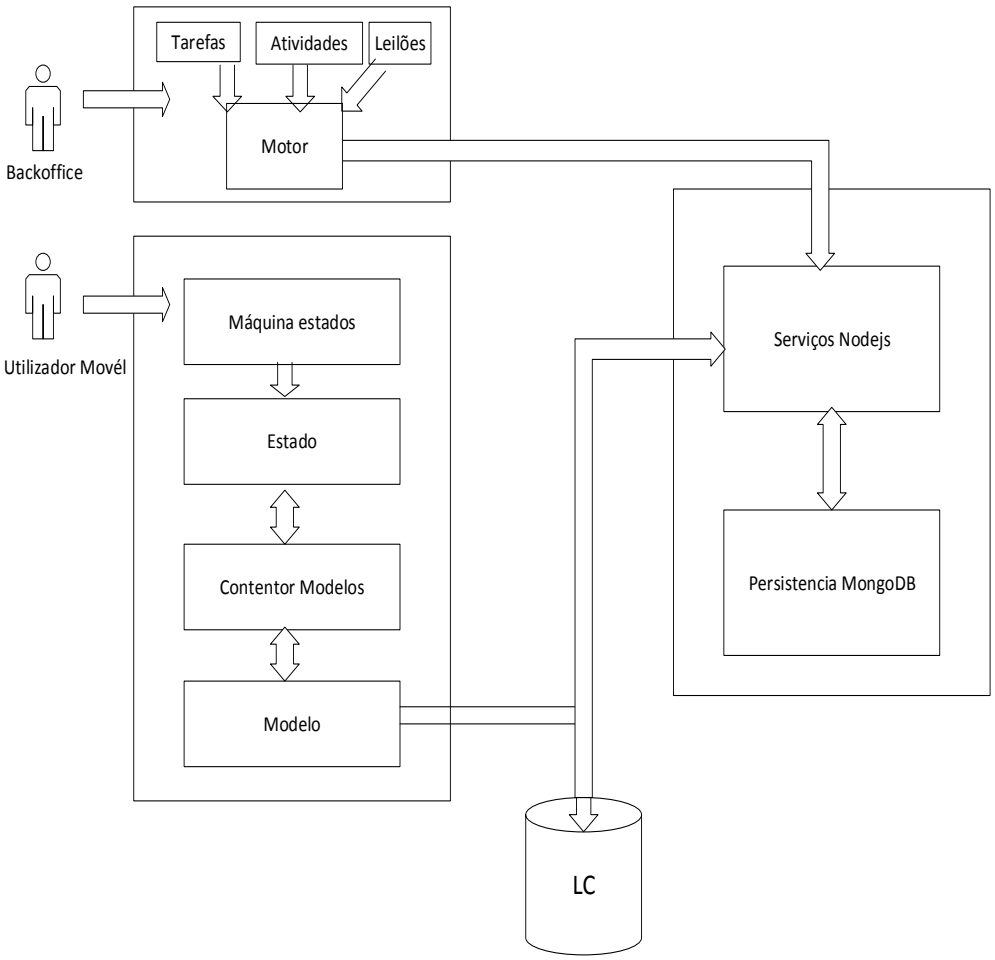


Figura 4.1 – Arquitetura da solução

A solução é composta por três módulos

- Motor de criação de tarefas
- *Framework* composta pela máquina de estados e pelo contentor de modelos
- Componente servidora responsável pelo lançamento dos leilões e persistência dos dados

4.1. *Framework* – Máquina de Estados e Contentor de Modelos

A figura 4.2 representa o diagrama de classes implementado para a máquina de estados.

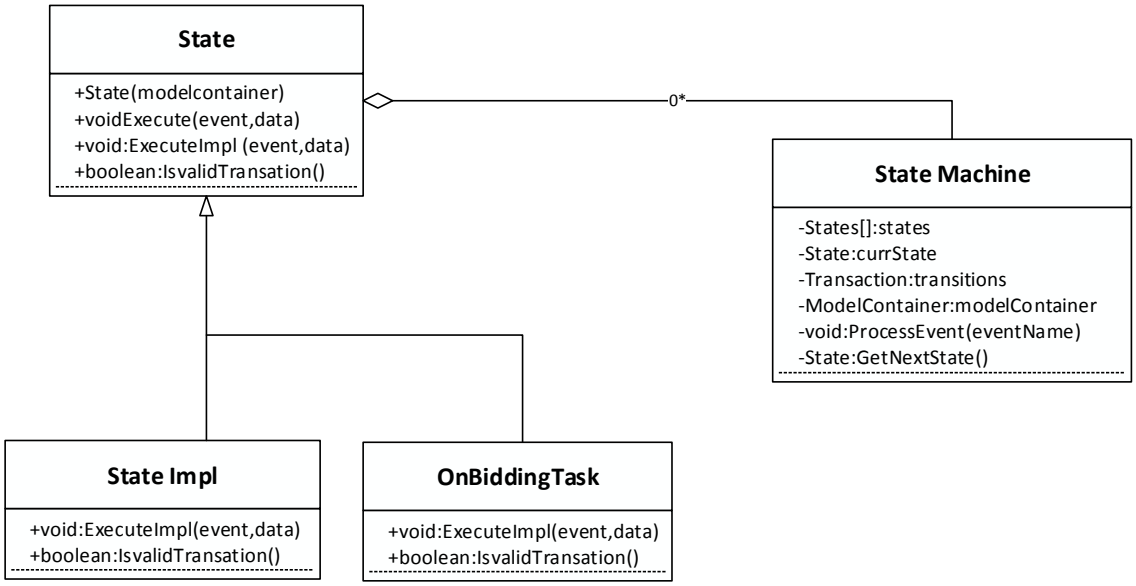


Figura 4.2– Diagrama de classes da máquina de estados

A implementação concreta de um objeto estado obriga a redefinição de dois métodos do objeto estado base

- *ExecuteImp*
- *IsValidTransaction*

O método *ExecuteImp* define a responsabilidade funcional do estado. É invocado pelo objeto base (*State*) na chamada ao método *Execute,* com o envio do evento despoletado e os respetivos dados associados. O método *Execute* é chamado pela máquina de estados na receção de eventos provenientes da interface gráfica.

No método *IsValidTransation* o estado valida a possível transição para um novo estado. Se por qualquer motivo a transição não possa ocorrer (e.g., dados errados ou incompletos introduzidos pelo utilizador) é da responsabilidade do próprio estado a sua indicação. Este método retorna um valor booleano e é chamado pela máquina de estados imediatamente antes da transição para um novo estado.

Cada estado é adicionado à máquina de estados de forma declarativa. Para isso, é necessário indicar no ficheiro configuração da máquina de estados o nome do objeto estado. Na sua criação, a máquina de estados consulta o ficheiro e agrega internamente todos os estados presentes. É também nesse momento que é iniciado o mecanismo *LongPolling* ao servidor para a verificação da conectividade ao longo do tempo.

Sempre que existe uma transição de estado, o estado corrente da máquina de estados é alterado. O novo estado é obtido através da chamada ao método *GetNextState*. Este método consulta a "tabela" de transição de estados (existente na máquina de estado) e devolve o novo estado, segundo o evento recebido e o estado em que se encontra.

A integração de estados é transparente para a implementação realizada. Os objetos *State* e o *StateMachine* mantêm-se inalteráveis.

A figura 4.3 representa o diagrama de classes implementada para o contentor de modelos.

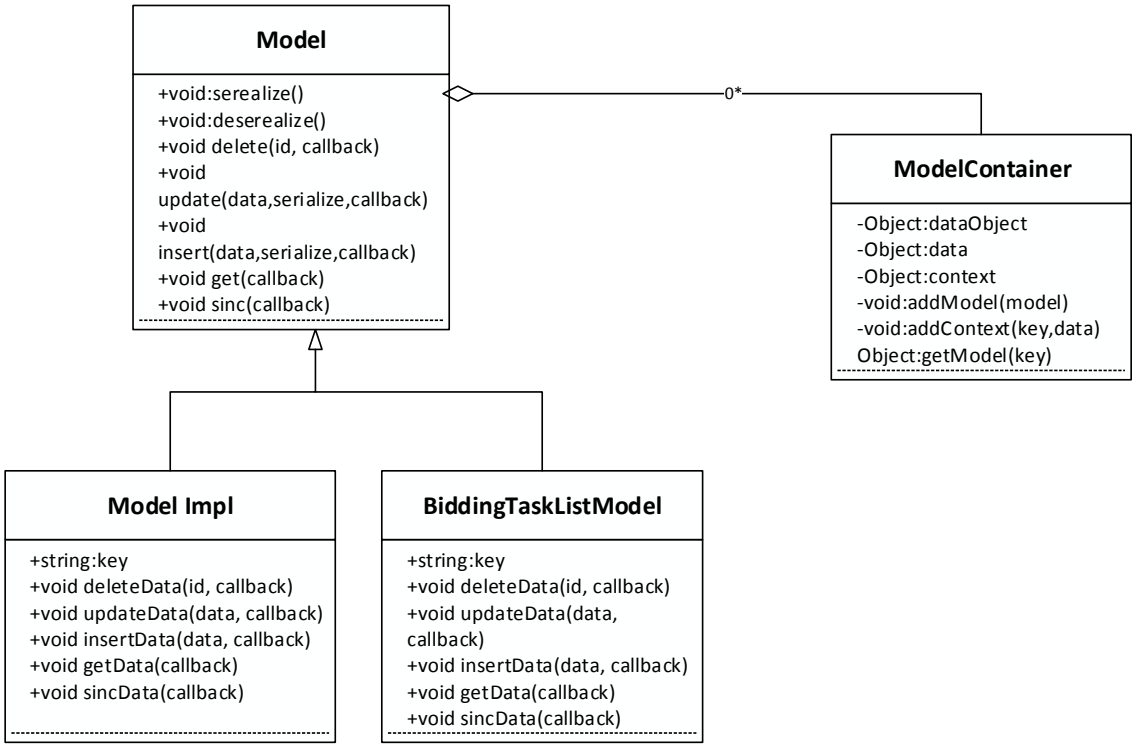


Figura 4.3 – Diagrama de classes do contentor de modelos

O objeto "Modelo" base (*Model*) personifica as operações *CRUD* através dos seguintes métodos

- *delete*
- *update*
- *insert*
- *get*

Estes métodos invocam a implementação concreta do modelo segundo o padrão "Operação_impl" (e.g., operação *update* - implementação do objeto modelo concreto *update_impl*).

Cada objeto "Modelo" é uma especialização de *Model* herdando todas as suas características. Destas características destaca-se o mecanismo de serialização e desserialização no *LocalStorage*. A chave identificadora do objeto no *LocalStorage* é indicada na propriedade *Key*. A mesma chave é utilizada para identificação no contentor de modelos.

Cada "Modelo" adiciona-se no contentor de modelos através de um processo de registo. O objecto *ModelContainer* (objeto representativo do contentor de modelos) fornece uma interface de registo através do método *addModel*. Para dados voláteis, objetos representativos do contexto de utilização, é disponibilizado o método *addContext*.

Como interface pública de acesso a estes objetos ("Modelo" e "Contexto") o contentor de modelos fornece os métodos *getModel e getContext*. Ambos os métodos recebem como parâmetro a chave de identificação do objeto.

A criação do contentor de modelos permite centralizar numa única entidade a informação relativa a dados diminuindo o acoplamento entre os objetos e consequentemente a diminuição de dependências entre os objetos. O mecanismo de *Binding* está presente em todos os objetos modelos. Este mecanismo é baseado no modelo *MVVM*. Foi incorporado na *Framework* através da biblioteca *Knockout* (18). A utilização deste padrão permite criar interfaces gráficas bastante apelativas através de código relativamente simples. A figura 4.4 demostra o troço de código utilizado para listar as tarefas atribuídas ao utilizador.

```
<div data-role="content">
    <ul data-role="listview" id="t2" data-divider-theme="e" data-inset="true">
        <li data-bind="delegatedClick: [{ EventFired: EVENT_SET.REFRESH}]" >As Minhas tarefas</li>
        <!-- ko foreach:  _moff_data.myTasks -->
        <li >
            <a data-bind="delegatedClick:[{ EventFired: EVENT_SET.SELECT_MYTASK}]" >
                <h3 data-bind="text: '#'+ id + ' | ' + category+ ' | '+ description "></h3>
                <span data-bind="text:taskvalue"  class='ui-li-count'></span>
            </a>
            <a  data-bind="delegatedClick: [{ EventFired: EVENT_SET.DISCARD_MYTASK}]">Descartar Tarefa</a>
        </li>
        <!-- /ko -->
    </ul>
</div>
```

Figura 4.4- Criação declarativa da interface gráfica

A ligação entre o controlo gráfico e o modelo de dados é realizada através do atributo *data-bind*. Para os objetos "Modelo" o *binding* segue a especificação "_moff_data_" concatenado com a chave identificadora do objeto "Modelo". Para os objetos contexto a especificação é "_moff_context" concatenado com chave identificadora do objeto "Contexto".

No exemplo demostrado na figura 4.4 a chave identificadora do objeto representativo da lista de tarefas é a *String* "*myTasks*" (especificação "*_moff_data.mytasks*"). A listagem das tarefas é obtida iterando sobre o objeto "Modelo" respetivo. Utilizando a função *foreach* da biblioteca *Knockout*, por cada elemento existente na coleção *myTasks* é criado dinamicamente um elemento *HTML* "<li>" com o identificador da tarefa, a categoria e a descrição.

O envio de eventos para a máquina de estados é realizado de forma declarativa segundo a especificação *"delegatedClick:[EventFired:eventoX]]"*. A máquina de estados implementa a interface "*delegatedClick"* redirecionando o evento para o estado corrente. Este mecanismo permite que o envio de eventos para a máquina de estados seja transparente para o programador.

4.2. BackOffice

É em *Backoffice* que se criam os leilões e as tarefas. Esta secção descreve a parametrização para a criação de leilões. É também especificada a arquitetura do motor de criação de tarefas caraterizando os módulos que o constituem.

Para a criação de um leilão é necessário indicar três parâmetros: o tipo de leilão, o tempo e o delta. Atualmente apenas é possível a criação de dois leilões, *First Price* e o

Vickerey. Não existe limite no número de leilões a serem criados. Sempre que existirem novas tarefas ou recepção de tarefas não efetuadas pelos utilizadores é possível a criação de um novo leilão. As tarefas não atribuídas num leilão estão disponíveis no leilão seguinte.

4.2.1. Motor de Criação Tarefas

O motor implementado é constituído por quatro módulos

- Tarefas – Módulo onde são descritas as tarefas e os seus objetivos
- Atividades – Módulo onde são criados os diferentes pressupostos
- Mapeamento – Módulo responsável pela associação das atividades às tarefas
- Motor (*Engine*) – Componente dinâmica que cria as tarefas com as respetivas atividades através do módulo de mapeamento

A criação de cada módulo é independente (diferentes ficheiros), podendo ser realizado por pessoas diferentes. Todos os módulos são representados em *JSON*. A figura 4.5 demostra a arquitetura do motor de criação de tarefas.

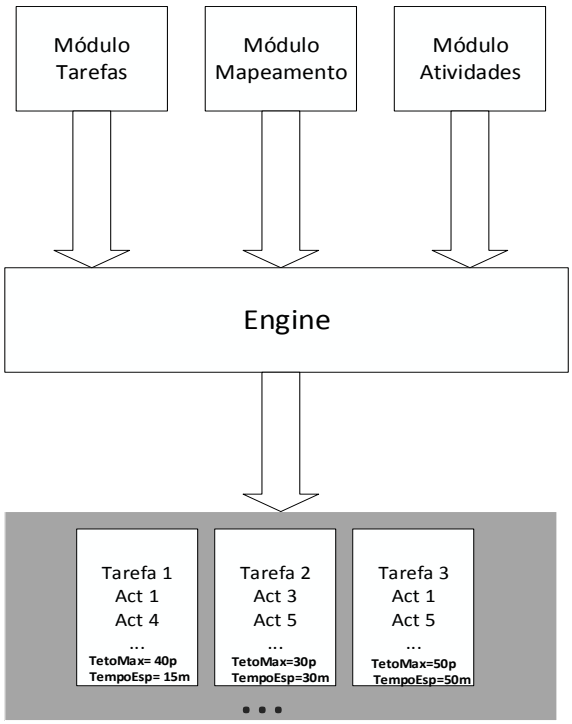


Figura 4.5– Arquitetura do motor de criação de tarefas

O motor recebe como entrada as tarefas, atividades e o módulo de mapeamento. Na saída são criadas as tarefas com as atividades mapeadas. A figura 4.6 apresenta o exemplo da criação do módulo de tarefas.

```
var tasks = [
                {description:'Auditoria para tecnico com a especialidade RC01',
                 category:'Auditorias em Redes'},
                {description:'Auditoria para tecnico com a especialidade RC02',
                category:'Auditorias em Redes'},
                {description:'Conteúdo programático Matemática 1º ciclo',category:'Matemática'},
                ...
            ]
```

Figura 4.6– Módulo de tarefas

Neste exemplo estão representadas três tarefas, duas relativas a auditorias a técnicos e outra relativa ao conteúdo programático de matemática.

Cada tarefa tem os seguintes campos

- *Description*: descreve o objetivo da tarefa
- *Category:* descreve o âmbito em que a tarefa está inserida

A figura 4.7 exemplifica a criação de três atividades relacionadas com avaliação do trabalho dos técnicos.

```
var activities =
[
    { id:1,question:"Têm equipamento de Medição", typeAnswer:'button', expectActivityTime:10000,
        possibleAnswers:[
        { idAnswer : 1,answer:"Sim"},
        { idAnswer : 2,answer:"Não"},
        { idAnswer : 3,answer:"S/R"}
        ]
    },
    { id:2,question:" Equipamento Calibrado", typeAnswer:'text', expectActivityTime:50000,
        possibleAnswers:[
        { idAnswer : 1,answer:"x<=0"},
        { idAnswer : 2,answer:"x>0 && x<10"},
        { idAnswer : 3,answer:"x>10"},
        { idAnswer : 4,answer:"S/R"}
        ]
    },
    { id:3,img:"equip.jpg",
    question:" Analisou corretamente o encaminhamento", typeAnswer:'button', expectActivityTime:20000,
    possibleAnswers:[
        { idAnswer : 1,answer:"Sim"},
        { idAnswer : 2,answer:"Não"},
        { idAnswer : 4,answer:"S/R"}
    ...
    ]
```

Figura 4.7- Módulo de atividades

Cada atividade é constituída pelos seguintes campos

- *Id*: identificador da atividade
- *question*: pressuposto a avaliar
- *typeAnswer*: tipo de resposta
    - *Button:* resposta fixa, constituída por um número variável de hipóteses
    - *Text:* expressão lógica avaliada com o parâmetro x = valor editado
- *expectActivityTime*: tempo esperado para a realização da atividade

- *img*: imagem representativa da atividade (opcional)

A Figura 4.8 representa o exemplo da criação módulo de mapeamento.

```
var mapping =
[
    {
    name:"RC01_map1",
    configuration : [
                    [1,1,2,5],
                    [1,2,3,5],
                    [1,3,3,0],

                    [2,1,3,10],
                    [2,2,3,10],
                    [2,3,3,10],
                    [2,4,3,0],

                    [3,1,4,5],
                    [3,2,4,5],
                    [3,3,4,5],
                    [3,3,4,0],
                    ....
    }
    ...
]
```

Figura 4.8 - Módulo de mapeamento

O módulo de mapeamento pode conter vários pacotes de configuração. Cada pacote é constituído

- *name*: nome representativo do pacote de configuração (a indicar ao motor no momento de criação)
- *configuration*: matriz de mapeamento

Cada tuplo da matriz de mapeamento tem seguinte representação

- Coluna 1: identificador da atividade
- Coluna 2: índice da resposta possível a ser dada pelo utilizador
- Coluna 3: identificador da próxima atividade segundo a resposta
- Coluna 4: pontos obtidos na resposta à atividade

É possível criar diferentes configurações com as mesmas atividades utilizando parametrizações diferentes (pontos obtidos,próxima atividade). A figura 4.9 apresenta o resultado da execução do motor segundo os parâmetros de entrada descritos anteriormente.

```
var result = [
            {description:'Auditoria para tecnico com a especialidade RC01',category:'Auditorias em Redes'
             activities:[
                {id:1,question:'Têm equipamento de Medição:',typeAnswer:'button',
                'expectActivityTime':10000,
                'possibleAnswers':[
                    {'idAnswer':1,'answer':'Sim','childId':2,'reward':5},
                    {'idAnswer':2,'answer':'Não','childId':3,'reward':5},
                    {'idAnswer':3,'answer':'S/R','childId':3,'reward':0},
                    ]
                },
                {id:2,question:'Equipamento Calibrado:',typeAnswer:'text',
                'expectActivityTime':50000,
                'possibleAnswers':[
                    {'idAnswer':1,'answer':'x<=0','childId':3,'reward':10},
                    {'idAnswer':2,'answer':'x>0 && x<10','childId':3,'reward':10},
                    {'idAnswer':3,'answer':'x>10','childId':3,'reward':10},
                    {'idAnswer':4,'answer':'S/R','childId':3,'reward':0},
                    ]
                },
                ...
                taskValue:20
                expectedTaskTime:80000
                ...
            ]
```

Figura 4.9 – Informação resultante do processamento do motor

Como resultado do processamento do motor as tarefas são aninhadas com atividades mapeadas. São criados igualmente dois novos campos

- *taskvalue*: valor correspondente ao valor de referência da tarefa, este valor é o teto máximo de licitação
- *expectedTaskTime*: tempo esperado para realização da tarefa

4.3. Servidor

O servidor disponibiliza serviços que permitem a persistência de dados no sistema *MongoDB*. Os serviços foram implementados segundo a arquitetura *REST*. A tabela 4.1 mostra os serviços implementados no servidor.

Verbo	URI	Descrição	Formato resposta
GET	auctionList	Devolve a informação de todos os leilões	JSON
GET	userList	Devolve todos os utilizadores registados na aplicação	JSON
GET	biddingtaskList	Devolve a lista de tarefas disponíveis para leilão	JSON
GET	task/:id	Devolve a informação de uma tarefa dado o seu *id*	JSON
GET	taskList/user/:id	Devolve as tarefas atribuídas a um utilizador dado o seu *id*	JSON
POST	bidding/:id	Cria uma licitação para uma tarefa	JSON
POST	sincOfflineBiddingTaskList	Atualiza as licitações realizadas em *Offline*	JSON
POST	sincOfflineTaskListResult	Atualiza as tarefas realizadas em *Offline*	JSON
POST	auction	Cria um novo leilão	JSON
POST	task	Cria uma nova tarefa	JSON

Tabela 4.1 – Descrição dos serviços implementados

Um pedido *REST* é efetuado através de um endereço único da mesma forma que fazemos para aceder a uma página *Web.* A diferença está no formato da resposta realizada pelo servidor. A página *Web* responde no formato *HTML* (*content-type=”HTML”*) enquanto para um serviço *REST* a resposta pode ser enviada em diferentes formatos. A figura 4.10 demostra a realização de um pedido *REST* que devolve a lista de utilizadores registados na aplicação. Para este pedido a resposta vem num formato *JSON* (*content-type=”JSON”*).

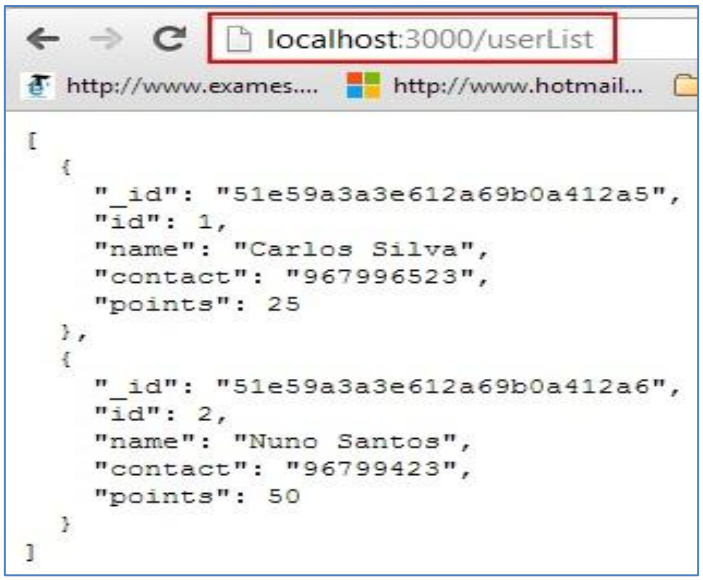


Figura 4.10 – Exemplo da invocação de um serviço *REST*

O *URI* fornecido tem o formato "Nome do Servidor / Nome do Serviço". Com o auxílio do *Browser* é possível de forma rápida validar a implementação dos serviços *REST* (apenas para os verbos *GET*).

4.3.1. Atribuição de Tarefas

A atribuição das tarefas negociadas pelos utilizadores em leilão é realizada no servidor. Além dos leilões *First Price* e *Vickrey*, a implementação efetuada suporta a possibilidade da incorporação de novos leilões. Para isso foi definido um contrato de implementação para a sua inclusão. A figura 4.11 demostra a solução adotada.

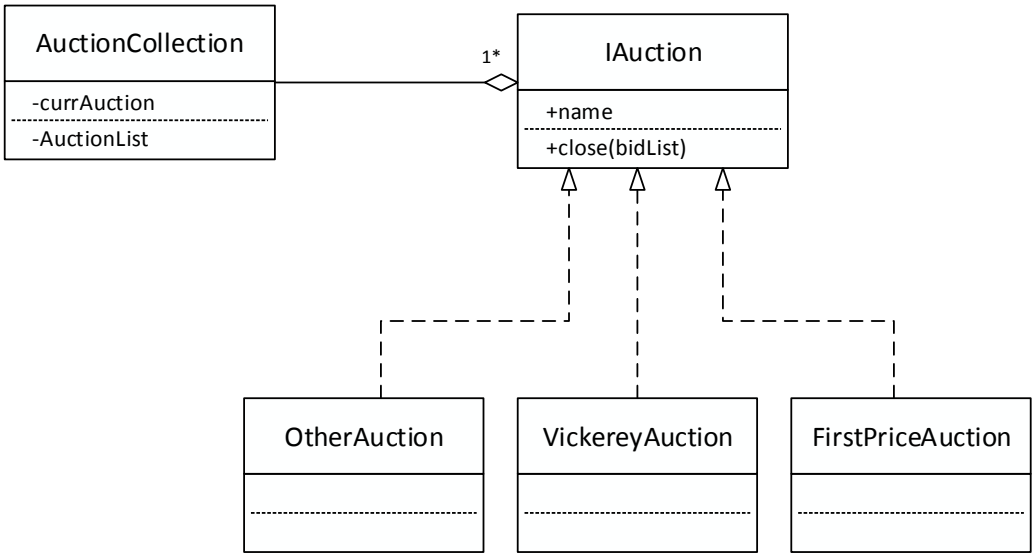


Figura 4.11 – Diagrama de classes do leilão

Para a criação de um novo tipo de leilão, será necessário a implementação da interface *IAuction*. Nesta interface, além do nome do leilão é obrigatório implementar o método *close* responsável pela afetação das tarefas aos vencedores. O método *close* recebe a lista de licitações recebidas para a tarefa, com a informação dos participantes e os respetivos valores de licitação.

De notar que a lista de licitações do método *close* apenas contêm uma licitação por utilizador. Os leilões implementados são leilões fechados onde cada utilizador realiza de forma privada uma licitação. Esta implementação foi realizada de modo a promover equidade entre utilizadores protegendo aqueles com conectividade intermitente ao servidor.

Uma nova implementação de um novo leilão podia ter em conta os pontos adquiridos pelos utilizadores (*ranking*) numa situação de "empate" no valor licitado. Atualmente o desempate é realizado segundo a data da licitação para os dois tipos de leilões implementados.

4.4. Navegação na Árvore de Tarefas

Para realização de uma tarefa o utilizador terá que responder a atividades. No início da realização da tarefa apenas é apresentado ao utilizador uma atividade. Esta atividade corresponde ao elemento raiz da árvore de atividades. À medida que o utilizador vai executando as atividades, vai navegando na árvore aparecendo-lhe dinamicamente as atividades correspondentes.

Uma das funcionalidades implementadas é a possibilidade da aplicação guardar diferentes árvores de navegação. Sempre que o utilizador modifica a resposta a uma atividade, e essa dá origem a uma outra atividade diferente da modificada, é quebrada a árvore de atividades até esse elemento. Neste cenário se por engano o utilizador modificasse uma atividade que provocasse a quebra da árvore perderia todo o trabalho efetuado anteriormente até esse elemento.

Para colmatar essa situação foi implementado uma funcionalidade "*friendly*" que permite ao utilizador a possibilidade de refazer as árvores anteriormente quebradas. A figura 4.12 demostra a funcionalidade implementada.

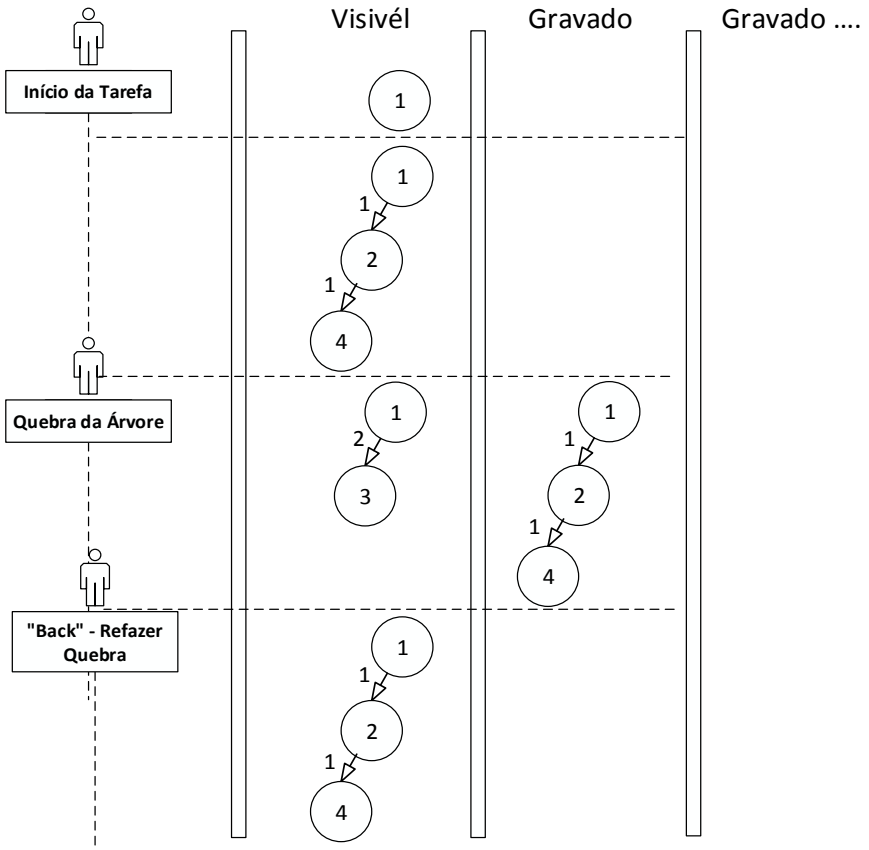


Figura 4.12 – Gravação de árvores de navegação

A área descrita como visível na figura corresponde às atividades que estão visíveis ao utilizador. As restantes áreas referem as diferentes árvores que foram desfeitas e gravadas em memória.

Inicialmente apenas é apresentado ao utilizador a atividade raiz da tarefa. De seguida o utilizador vai realizando atividades até quebrar a árvore anteriormente construída. Essa quebra foi despoletada pela alteração da resposta à atividade 1. Uma vez que a resposta 2 à atividade 1 dá origem a uma atividade diferente da resposta 1 toda a árvore construída até essa atividade é desfeita. No entanto a árvore é gravada em memória para que o utilizador a possa repor.

Através da funcionalidade "*Back*" o utilizador repõe a última árvore de atividades quebrada, destruindo a visível. A figura 4.12 apresenta apenas uma situação de gravação de uma árvore de atividades, no entanto poderá ser guardado mais que uma árvore sendo resposto sempre a última árvore desfeita.

Esta funcionalidade permite ao utilizador navegar entre a árvore de atividades presente na tarefa dando-lhe a possibilidade de refazer alguns erros cometidos, aumentando o seu grau de confiança na execução das suas tarefas.

5. Validação e Testes

Neste capítulo é apresentada a validação realizada para a aplicação móvel implementada. A validação efetuada é centrada no utilizador com a utilização de testes de usabilidade. Os testes de usabilidade são um método de avaliação de sistemas interativos que recorrem à utilização de questionários ou entrevistas com o utilizador de forma a avaliar o funcionamento do sistema.

5.1. Objetivos

A comunidade científica colocou à disposição vários questionários dedicados à avaliação de usabilidade. Apesar de amplamente estudados, o objetivo de generalização dos questionários a todos os casos não permite por vezes o estudo de casos.

Assim numa primeira fase foi realizado um estudo dos questionários disponíveis (19) numa tentativa de seleção de perguntas a integrar no questionário. Posteriormente foram criadas novas perguntas dirigidas especificamente às funcionalidades da aplicação.

O questionário criado (Anexo I) visa avaliar três vectores

- Avaliação da interface gráfica do utilizador
- Avaliação do funcionamento geral do *Software*
- Avaliação de funcionalidades específicas do *Software*

Cada vector apresentado corresponde a um grupo do questionário. Cada grupo é constituído por cinco perguntas. Cada pergunta é respondida numa escala crescente de 1 a 5 (1- fraco, 5- muito bom).

5.2. Resultados

Foram realizados oito inquéritos a pessoas com idades compreendidas entre os 12 e os 65 anos. Uma vez que a aplicação foi implementada de forma a suportar diferentes domínios do problema era importante distribuir os participantes por diferentes faixas etárias. A tabela 5.1 demostra os resultados obtidos.

		Grupo I						Grupo II						Grup III						
S	Idade	1.1	1.2	1.3	1.4	1.5	%	2.1	2.2	2.3	2.4	2.5	%	3.1	3.2	3.3	3.4	3.5	%	Total %
M	14	4	4	4	3	5	80	5	4	4	4	4	84	5	4	5	5	4	92	85,333
M	15	4	3	4	3	4	72	5	5	4	3	4	84	5	4	5	5	4	92	82,667
F	30	4	4	4	4	5	84	4	5	4	4	4	84	4	4	5	4	4	84	84
F	33	5	4	4	4	4	84	4	4	4	3	4	76	4	4	5	4	4	84	81,333
M	42	4	4	3	3	3	68	3	4	4	4	3	72	3	4	4	4	4	76	72
M	50	5	4	4	3	3	76	4	4	4	3	3	72	4	4	4	4	4	80	76
F	55	4	3	4	3	4	72	4	4	4	3	3	72	3	4	5	5	4	84	76
F	65	4	3	3	4	3	68	4	4	4	4	3	76	3	3	4	4	4	72	72
Totais 14-35							80						82						88	83,333
Totais 36-65							71						73						78	74
Totais							76						78						83	78,667

Tabela 5.1 – Tabela de resultados aos inquéritos realizados

A partir da amostra utilizada é possível retirar algumas conclusões aos inquéritos realizados.

O grau de satisfação da aplicação é positivo, cerca de 78,7%. O grupo de questões que obteve melhores resultados (83%) foi relativo a funcionalidades específicas da aplicação ficando demonstrada uma satisfação geral relativamente às mesmas.

Os utilizadores de menor idade revelaram uma maior receptividade (82.3%) da aplicação relativamente aos utilizadores de maior idade (74%). Essa situação talvez possa ser explicada pelo facto de a aplicação funcionar num dispositivo móvel. Os mais novos talvez estejam mais familiarizados com estes dispositivos sentindo menor dificuldade na sua utilização.

6. Conclusões

A implementação de aplicações *Web-Offline* pode ser concretizada através da utilização de plataformas proprietárias ou através do *Browser* como plataforma de execução. Este projeto implementa uma infraestrutura de desenvolvimento e duas aplicações concretas (suportadas nessa infraestrutura) apenas baseadas na utilização d*o Browser*.

Com esta abordagem não existe, para além do *Browser*, qualquer dependência tecnológica; a implementação é integralmente realizada em *Javascript* (linguagem interpretada pelo *Browser*). Assim a solução não está comprometida com especificidades de plataformas proprietárias que apenas poderão ser suportadas pelos próprios fornecedores.

Esta solução também está completamente desacoplada à componente servidora. A função servidora apenas fornece dados pelo que uma mudança de tecnologia no lado do servidor não implica qualquer mudança na implementação realizada no cliente.

A *Framework* implementada permite dar aos programadores um padrão de raciocínio baseado na divisão de responsabilidades funcionais em estados. A utilização da máquina de estados em ambiente *Web* permite uma validação assertiva das ações do utilizador no lado do cliente, evitando a utilização da interface gráfica para validações. A grande maioria das *Web Applications* recorre à inibição de controlos gráficos do *Browser* como forma de evitar ações do utilizador. O problema é que os *Browsers* têm implementações diferentes (não seguem a especificação), causando diferentes comportamentos nas suas interfaces gráficas.

A *Framework* fornece ainda um conjunto de funcionalidades que poderão ser reutilizadas noutros projetos neste âmbito. Dessas funcionalidades destaca-se a capacidade de *Binding* automático com a interface gráfica e capacidade de persistência local.

A utilização do leilão como mecanismo de afetação de recursos permite ao utilizador escolher as tarefas que lhe são mais convenientes. É uma boa solução para cenários móveis uma vez que o utilizador pode optar por tarefas mais próximas do local em que

se encontra. O leilão fechado pretende proteger utilizadores com fraca conectividade, tentando promover uma equidade entre todos os participantes.

O motor de criação de tarefas é uma implementação genérica que foi construído de forma a poder ser utilizado noutras implementações. Tem uma componente de configuração flexível com possibilidade de criação de questionários em formato de resposta fixa ou editável. Os questionários podem ter uma estrutura sequencial ou em árvore (em caso de dependência entre perguntas).Neste projeto o motor implementado foi utilizado em dois domínios diferentes (auditorias e conteúdos pedagógicos), existem no entanto outros domínios onde pode ser aplicado (e.g., inquéritos de satisfação, testes de usabilidade de aplicações).

7. Trabalho Futuro

A possibilidade de serialização local utilizando outros tipos de *Storages* existentes seria uma forma de tornar a *Framework* mais completa.

O *LocalStorage* foi a tecnologia escolhida devido à sua simplicidade e por ser implementada pela maioria dos *Browsers*. Possui no entanto algumas fragilidades nomeadamente no volume de informação que consegue alojar (cerca de 4 *MegaBytes*). Existem implementações mais complexas mas simultaneamente mais completas como o *Web SQL Database* onde é possível interrogar os dados utilizando a linguagem SQL.

A criação de um editor gráfico integrado no motor de criação de tarefas seria também uma implementação enriquecedora. As tarefas e as atividades são criadas declarativamente utilizando código *JSON*. A implementação de uma interface gráfica que gerasse o código JSON permitia a criação rápida das tarefas e reduzia erros na sua criação.

Bibliografia

1. A Beginner's Guide to the Different eBay Auction Types. [Online] [Cited: 6 1, 2013.] http://www.ebay.com/gds/A-Beginner-apos-s-Guide-to-the-Different-eBay-Auction-Types-/10000000004003874/g.html.

2. **Adamski, Lucas.** Adamski. Introducing Adobe AIR security model. [Online] 2008.

3. **Chris, Loosley.** Rich Internet Applications:Design, Measurement,and Management Challenges. [Online] 2006. [Cited: 6 1, 2013.] http://www.keynote.com/docs/whitepapers/RichInternet_5.pdf.

4. Mozilla Labs. [Online] 8 2007. [Cited: 6 1, 2013.] http://labs.mozilla.com/2007/10/prism/.

5. **Papa, John.** Building An Out-of-Browser Client With Silverlight 3. [Online] 6 2009. [Cited: 6 1, 2013.] http://msdn.microsoft.com/en-us/magazine/dd882515.aspx.

6. **Almear, Dion.** How to take your Web Application Offline with Google Gears. [Online] 10 3, 2007. [Cited: 6 1, 2013.] http://www.slideshare.net/dion/future-of-web-apps-google-gears.

7. **Pilgrim, Mark.** LET'S TAKE THIS OFFLINE. [Online] [Cited: 6 1, 2013.] http://diveintohtml5.info/offline.html.

8. **Jim , Webber, Robinson, Ian and Savas , Parastatidis.** REST in Practice. [Online] Rachel Monaghan, 9 2010. [Cited: 5 1, 2013.] http://it-ebooks.info/book/393/.

9. **Smith, Josh .** WPF Apps With The Model-View-ViewModel Design Pattern. [Online] 2 2009. [Cited: 4 1, 2013.] http://msdn.microsoft.com/en-us/magazine/dd419663.aspx.

10. **Gundavaram , Shishir .** CGI Programming on the World Wide Web. [Online] O'Reilly Open Books Project, 3 2006. [Cited: 10 1, 2012.] http://oreilly.com/openbook/cgi/ch05_01.html.

11. **Hertel, Matthias.** Aspects of AJAX. [Online] 1.2, 2007. http://www.mathertel.de/Ajax/AspectsOfAJAX0704.pdf.

12. **Takada, Mikito.** Single page apps in depth. [Online] [Cited: 3 1, 2013.] http://www.nuff.ox.ac.uk/users/klemperer/VirtualBook/07_Chapter1.pdf.

13. **Klemperer, Paul.** *Auctions: Theory and Practice.* s.l. : Princeton University Press and copyrighted, 2004.

14. **Hickson, Ian.** The WebSocket API. [Online] 7 2013. http://dev.w3.org/html5/websockets/.

15. **Russel, Alex.** Low latency data for the browser. [Online] [Cited: 8 1, 2013.] http://alex.dojotoolkit.org/?p=545,.

16. **Vaish, Gaurav.** Getting Started with NoSQL. [Online] Packt Publishing, 2013. [Cited: 4 1, 2013.] http://it-ebooks.info/book/2748/.

17. **Seguin, Karl.** The Little MongoDB Bool. [Online] [Citação: 1 de 6 de 2013.] http://openmymind.net/mongodb.pdf.

18. knockout. *Simplify dynamic JavaScript UIs with the Model-View-View Model (MVVM) pattern.* [Online] [Citação: 1 de 2 de 2013.] http://knockoutjs.com/.

19. Software Usability Measurement Inventory. [Online] http://sumi.ucc.ie/.

Anexos I – Inquérito de Satisfação

Inquérito de avaliação de Usabilidade

Nome:

Idade:

Sexo:

1 2 3 4 5

Grupo I

Avaliar a satisfação do utilizador relativamente ao uso da interface

1.1 - A interface gráfica apresentada é atrativa

1.2- Os menus estão organizados de forma lógica

1.3- A utilização da aplicação é fácil e intuitiva

1.4- As mensagens de texto são esclarecedoras

1.5- É fácil de navegar na aplicação

Grupo II

Avaliar a funcionalidade do sistema

2.1- A aplicação é robusta

2.2- Sente confiança na utilização desta aplicação

2.3- A aplicação responde de forma rápida ao utilizador

2.4- É necessário efetuar poucos passos para a realização das funcionalidades

2.5- Utilizaria esta aplicação todos os dias

Grupo III

Identificação de problemas específicos do funcionamento geral do sistema

3.1-O processo de licitação é simples

3.2- A forma de afetação de recursos é adequada

3.3- A verificação do estado da conectividade é uma boa funcionalidade

3.4- A funcionalidade “back” permite dar mais confiança ao utilizador na utilização da aplicação

3.5- Concorda com o processo de atribuição de pontos ao utilizador