UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

DANIEL HOFFMANN
TATIANA ROMANOVSKI GUILL MORAES

ULIXES, UM APLICATIVO DO TIPO *FIELD SERVICE MANAGEMENT SYSTEM*

CURITIBA
2011

DANIEL HOFFMANN

TATIANA ROMANOVSKI GUILL MORAES

ULIXES, UM APLICATIVO DO TIPO *FIELD SERVICE MANAGEMENT SYSTEM*

Monografia de Conclusão de Curso apresentada ao Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná como requisito para obtenção do grau de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Msc. Luiz Augusto Pelisson.

CURITIBA

2011

Ficha Catalográfica

DANIEL HOFFMANN

TATIANA ROMANOVSKI GUILL MORAES

ULIXES, UM APLICATIVO DO TIPO *FIELD SERVICE MANAGEMENT SYSTEM*

Esta monografia foi julgada e aprovada para a obtenção do grau de **Tecnólogo** no **Programa de Graduação** em Tecnologia em Sistemas para Internet
da Universidade Tecnológica Federal do Paraná.

Curitiba, 29 de Setembro de 2011.

Prof. Wania Meira Matos Figueredo
Coordenadora do Curso de Tecnologia em Sistemas Para a Internet

BANCA EXAMINADORA

Prof. Luiz Augusto Pelisson
Universidade Tecnológica Federal do Paraná
Orientador

Prof Marcelo Mikosz Gonçalvez
Universidade Tecnológica Federal do Paraná

Prof Ana Cristina Barreiras Kochem Vendramin
Universidade Tecnológica Federal do Paraná

RESUMO

HOFFMANN, D.; MORAES, T. R. G. **Ulixes, um Aplicativo do Tipo Field Service Management System.** 2011. Monografia (Graduação em Tecnologia em Sistemas para Internet) – Departamento Acadêmico de Informática (DAINF), UTFPR, Curitiba.

A complexidade inerente ao processo de gerenciamento de serviços de campo prestados pelas empresas levou ao desenvolvimento de diversos sistemas com o propósito de organizar tal fluxo de atividades. Esses aplicativos, de uma maneira geral, respondem de forma insatisfatória aos requisitos dos usuários. Esse trabalho se dispõe, baseado em um amplo levantamento de requisitos, a atender mais completamente tais necessidades. Para isso, utilizar-se-á um novo paradigma de desenvolvimento de sistemas, o RIA (Aplicações de Internet Rica), que alia uma ótima experiência de utilização com a facilidade de manutenção e atualização das ferramentas *Web*, com a intenção de construir um novo aplicativo ao final do processo que sintetizará tais requerimentos.

Palavras-Chave

Sistemas de Gerenciamento de Força de Campo, Aplicações de Internet Rica, Softwares de Aplicação.

ABSTRACT

HOFFMANN, D.; MORAES, T. R. G. **Ulixes, um Aplicativo do Tipo Field Service Management System.** 2011. Monografia (Graduação em Tecnologia em Sistemas para Internet) – Departamento Acadêmico de Informática (DAINF), UTFPR, Curitiba.

The inherent complexity in the process of managing field services provided by a company led to the development of several systems in order to organize such activity flow. These applications, in general, respond unsatisfactorily to the user requirements. This work intends, based on a broad survey of requirements, to meet those needs more completely. For this, will be used a new paradigm for systems development, the RIA (Rich Internet Applications), which combines an excellent user experience with ease of maintenance and updating of Web tools, with the intention to build a new application at the end of the process that summarizes those requirements.

Key-words

Field Service Management Systems, Rich Internet Applications, Application software.

LISTA DE SIGLAS

AT -*Attention Terminal*
CEP -Código de Endereçamento Postal
CNPJ -Cadastro Nacional de Pessoa Jurídica
CPF -Cadastro de Pessoas Físicas
CRM -*Customer relationship management*
DAO -*Data Access Objects*
EE -*Enterprise Edition*
EJB -*Enterprise Java Bean*
FSM - *Field Service Management*
GB - *Gigabyte*
GSM - *Global System for Mobile Communications*
HTML - *HyperText Markup Language*
HTTP - *Hypertext Transfer Protocol*
IDE - *Integrated Development Environment*
JDK - *Java Development Kit*
JNI -*Java Native Interface*
JVM - *Java Virtual Machine*
LCDS - *LifeCycle Data Services*
MVC -*Model View Control*
MXML- *Magic Extensible Markup Language*
RAM - *Random Access Memory*
RIA -*Rich Internet Applications*
SGBD - Sistema de Gerenciamento de Banco de Dados
SMS - *Short Message Service*
SSH - *Secure Shell*
USB - *Universal Serial Bus*
WORA-*Write Once, Run Anywhere*
XAML- *Extensible Application Markup Language*
XML - *Extensible Markup Language*

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

O projeto aqui apresentado modela e implementa um sistema de gerenciamento de força de campo. O sistema referido será responsável por armazenar dados de solicitações técnicas dos clientes, permitindo o ajuste e recuperação destes dados e controlando o fluxo de atendimento de tais solicitações.

1.1 JUSTIFICATIVA

O fluxo das atividades que compõem os serviços de campo de uma empresa reúne uma complexidade suficiente para que seja mais vantajoso permitir o seu gerenciamento por um sistema. É necessário guardar todos os dados envolvidos de forma segura, para que o atendimento possa ser prestado ao cliente no local e prazo adequados, e para que cada técnico de campo saiba o que deve ser feito. A empresa precisa também manter um histórico das solicitações de serviço — os *tickets* — tanto para remunerar corretamente os responsáveis quanto por razões administrativas.

A empresa tomada por base para a elaboração desta proposta possui atualmente um sistema responsável pelo desempenho de parte destas funções; porém, tanto por falhas técnicas quanto por um levantamento incompleto de requisitos, este aplicativo não tem suprido satisfatoriamente as necessidades da companhia, gerando impacto no atendimento aos clientes e causando problemas na base de dados utilizada pelo software.

1.2 OBJETIVOS

As metas que motivaram a concepção do Ulixes podem ser dividas em um objetivo geral e em um conjunto mais amplo de objetivos específicos.

1.2.1 Objetivo Geral

O objetivo do projeto é produzir um sistema capaz de suprir a necessidade das empresas de organizar e gerenciar o serviço feito em campo pelos técnicos das mesmas. Várias empresas já possuem aplicativos destinados a tal propósito, contudo o Ulixes se propõe a atender mais completamente os requisitos delas nesse âmbito.

1.2.2 Objetivos Específicos

Os objetivos específicos do Ulixes são:

- Prover uma interface gráfica agradável ao usuário e com baixa curva de aprendizado;
- Permitir a extração de relatórios;
- Permitir correções nas informações das solicitações;
- Realizar o envio de solicitações de atendimento aos técnicos, com agilidade;
- Organizar os dados referentes aos colaboradores da empresa, aos clientes e aos atendimentos prestados, de maneira clara;
- Manter estes dados armazenados de modo seguro e consolidado (a informação não pode estar dispersa nos diversos computadores dos usuários);
- Controlar os acessos para que apenas usuários autorizados possam ver certas informações (como cadastros de cliente) e realizar operações dentro do sistema;
- Oferecer um sistema que possa ser facilmente adaptado para outros Sistemas Gerenciadores de Bancos de Dados, para vários navegadores e sistemas operacionais;
- Oferecer um sistema preparado para ser expandido com o acréscimo de novas funcionalidades, caso se mostre necessário;
- Facilitar a manutenção do sistema, com a apresentação de algumas telas *CRUD* (*Create, Read, Update and Delete*) para intermediar o acesso aos dados tanto dos usuários quanto dos perfis disponíveis na aplicação.

1.3 ESCOPO

Respeitando os objetivos levantados, a aplicação desenvolvida é acessível via rede, e permite aos usuários conectarem-se a ela através de um sistema de autenticação. Ela oferece uma interface para que o administrador do Ulixes na empresa cliente possa cadastrar acesso para novos usuários, e também fazer a alteração dos mesmos quando necessário – um recurso que se enquadra na já mencionada facilidade de manutenção da aplicação.

O menu de funcionalidades da aplicação é customizável, alterando-se para refletir o perfil de permissões do usuário que está acessando. As ações do usuário são registradas em uma tabela de log. Estas duas características foram implementadas para atender o quesito de segurança.

O sistema permite a inserção de solicitações de serviço por interface gráfica e via *web service*, e os dados da solicitação são pesistidos de forma confiável. A alteração de solicitações pode ser feita via *web services* (para que o status do ticket seja atualizado por outros processos da empresa) e através de interface gráfica.

Com antecedência de uma hora do horário agendado para atendimento, a aplicação faz o envio automático da solicitação para o dispositivo móvel do técnico eleito como responsável.

Para resumir, estão no escopo do projeto:

- A capacidade de manipular os dados de *tickets*, clientes, empreiteiras e técnicos;
- O controle de acesso aos recursos da aplicação;
- O registro de operações realizadas através de log;
- O envio de solicitações a um dispositivo móvel previamente cadastrado no Ulixes.

Não fazem parte do escopo do Ulixes:

- Envio de *tickets* ou outros tipos de mensagem SMS a dispositivos móveis cujos números não constem na base de dados do sistema;
- Acesso a outros *softwares* (o Ulixes disponibiliza *web services* para que aplicações externas se conectem a ele, mas ele próprio não acessa sistemas de CRM ou outros programas externos);
- Oferecer interfaces para que o cliente de um *ticket* utilize diretamente o sistema – o Ulixes foi concebido para o uso de colaboradores da empresa prestadora de serviços, e estes funcionários estarão intermediando a comunicação entre cliente externo e aplicação.

O Ulixes e seus desenvolvedores também não são de modo algum responsáveis pelas soluções providas pelos técnicos, quando estes estão prestando suporte ao cliente; o sistema não oferece ferramentas de diagnóstico de problemas ou repositório de informações técnicas – seu papel é exclusivamente o de gerenciar dados de atendimento.

1.4 PÚBLICO ALVO

O público alvo do Ulixes é a empresa cujo cenário foi estudado para a concepção do sistema. Os usuários da aplicação serão os colaboradores da empresa, notavelmente analistas de negócio, analistas de suporte, técnicos de campo e seus supervisores.

Estes usuários já têm alguma experiência com *Field Service Management Systems*, uma vez que a companhia possui um sistema do gênero instalado. Ainda assim, pretende-se criar uma interface intuitiva que facilite a migração para o Ulixes, proporcionando uma curva de aprendizado mais eficiente e possibilitando o uso da aplicação mesmo por usuários leigos.

1.5 RESTRIÇÕES E RISCOS

As restrições para o funcionamento do Ulixes são poucas: o sistema demanda que o servidor possua instalado o *Java Development Kit* (JDK) versão 6 ou superior, e também o SGBD Oracle (embora não seja difícil modificar a aplicação para funcionar com outro banco de dados, em caso de necessidade). Os computadores dos usuários deverão ter acesso ao servidor; também precisam possuir um navegador web instalado, e o *plugin* Adobe Flash Player versão 10 ou superior, para este mesmo navegador.

Uma vez implementado o sistema, o primeiro e principal risco é que o BlazeDS – solução de código aberto utilizada em partes do Ulixes (para informações mais detalhadas, consultar o tópico “Tecnologias Utilizadas”) – não apresente um desempenho satisfatório com o crescimento exponencial da base de usuários. De acordo com a experiência dos integrantes do projeto, o BlazeDS não possui as otimizações necessárias para lidar com um volume muito grande de dados. Por outro lado, a migração do sistema para o *LifeCycle Data Services*, produto comercial equivalente ao BlazeDS, é muito simples uma vez que ambos são compatíveis, o que ameniza bastante a preocupação com escalabilidade a longo prazo.

O outro risco avaliado relaciona-se justamente com as restrições apresentadas no início do tópico. Como parte do processamento do Ulixes ocorre na máquina do usuário, dentro do Flash Player ali instalado, um computador com processador muito antiquado ou memória RAM insuficiente pode apresentar severas limitações de desempenho ao acessar o sistema. O Ulixes não tem pré-requisitos muito elevados para funcionar (foi extensivamente testado em um *notebook* com processador Dual Core de 2.3 GHz, com 3 GB de memória RAM, e consumiu uma quantidade de recursos muito pequena durante o uso), mas pode ter uma performance pouco satisfatória em máquinas demasiadamente obsoletas.

1.6 METODOLOGIA.

O desenvolvimento do Ulixes ocorreu de maneira iterativa e incremental.

Tomando por base os requisitos levantados, foram esboçadas as funcionalidades essenciais. Implementadas na forma de protótipos, estas foram testadas e quaisquer erros, problemas de usabilidade ou incompatibilidade com os requisitos, solucionados na seqüência. À medida que estas ferramentas alcançavam um grau satisfatório, outras eram abordadas, partindo-se dos casos de uso críticos para os úteis ou desejáveis.

Acompanhando esse processo de evolução gradativo, a documentação sofreu contínuos ajustes para refletir o amadurecimento do sistema.

Consideramos este método de trabalho bastante satisfatório, pois uma metodologia mais rígida não apresentaria o desembaraço e a rapidez necessária para a conclusão do Ulixes dentro do tempo disponível. Os preceitos de desenvolvimento ágil, uma outra resposta possível para o problema da falta de tempo, não eram adequados ao projeto, pois foram concebidos para a produção de *software* com pouca ou nenhuma documentação – e desejava-se que o Ulixes fosse documentado de maneira bastante completa. Logo, o modelo em espiral revelou-se o mais adequado.

A única dificuldade que foi percebida ao se utilizar este procedimento é a determinação do momento certo para se considerar uma funcionalidade concluída. É uma tentação, principalmente para desenvolvedores de caráter perfeccionista, continuar a aprimorar a mesma ferramenta *ad infinitum*, ao invés de contentar-se com um nível razoável de qualidade e prosseguir para a ferramenta seguinte.

1.7 ESTRUTURA DO DOCUMENTO

Este subtópico destina-se a falar não da estrutura do *software* – que é pormenorizada em páginas posteriores – e sim da estrutura deste documento. O capítulo 1 tem como objetivo apresentar o projeto de modo geral, mostrando suas motivações, descrevendo o que o Ulixes propunha-se ou não a oferecer, e esboçando a maneira como foi desenvolvido.

O capítulo 2 do documento estabelece algumas bases teóricas ao abordar as RIA (*Rich Internet Applications*), e o panorama de seu surgimento. Sem o conceito de RIA e as várias tecnologias criadas para torná-lo concreto e alcançável, não teria sido possível implementar o sistema Ulixes.

O capítulo 3 aborda o sistema propriamente dito. O tópico começa detalhando os requisitos identificados. Em seguida, discorre sobre as tecnologias empregadas de modo técnico, justificando a escolha das mesmas para a execução do projeto. Peculiaridades destas tecnologias são rapidamente descritas, com o intuito de facilitar a compreensão da arquitetura do sistema. As soluções consideradas pelos desenvolvedores como mais interessantes ou criativas são destacadas em um subtópico próprio, e então parte-se para uma extensa exposição da aplicação.

O sistema Ulixes é explicado com o auxílio de diagramas UML e textos auxiliares. As diversas funcionalidades são enumeradas em diagramas de caso de uso, e a estrutura do banco de dados é apresentada em um Modelo Entidade-Relacionamento. A documentação que representa mais claramente os mecanismos internos da aplicação, contudo, são os diagramas de seqüência, que podem ser encontrados no apêndice.

Finalmente, o capítulo 4 é uma compilação de lições aprendidas e conclusões derivadas deste projeto; ele é seguido pelas referências consultadas e pelo apêndice.

2 SOFTWARES DE APLICAÇÃO E WEB RIA

A indústria de *software* sempre primou pelo desenvolvimento de sistemas que diminuíssem os custos de operação dos negócios para os quais foram projetados. Em geral, a demanda pela automatização de uma determinada tarefa pela introdução de um sistema computacional justificava-se pela diminuição de custos relacionados a contratação de funcionários e a compensação por falhas geradas pela operação de sistemas manuais.

Devido a tais exigências, os softwares criados com a utilização deste paradigma se tornaram muito robustos e eficientes, contudo, deixando muito a desejar no quesito usabilidade. Todas as tentativas de se produzir sistemas que proporcionassem uma melhorar experiência de utilização para o usuário acabam por gerar sistemas mais complexos e mais onerosos em sua manutenção.

Nesse contexto surgem um novo tipo de paradigma, que busca aliar as interfaces ricas com a facilidade de manutenção, as chamadas aplicações RIA (*Rich Internet Applications*).

Neste capítulo serão abordados os softwares de aplicação sob uma perspectiva generalista, as aplicações *web* e o paradigma RIA de desenvolvimento de sistemas, que será aplicado em nosso trabalho para o desenvolvimento de uma software de aplicação.

2.1 SOFTWARES DE APLICAÇÃO

Os softwares desenvolvem papel crucial na melhoria dos processos empresariais, reduzindo custos e melhorando a qualidade do serviço prestado por tais empresas. Esses softares, voltados a resolução de problemas específicos de um determinado negócio, são chamados de *softwares* de aplicação.

Pressman (2006, p. 6) define um *sofware* de aplicação como sendo:

> "[...] programas isolados que resolvem uma necessidade especifica do negócio. Aplicações nessa área processam dados comerciais ou técnicos de um modo que facilita as operações ou gestão/tomada de decisões técnicas do negócio. Além das aplicações convencionais de processamento de dados, o software de aplicação é usado para controlar funções do negócio em tempo real (por exemplo, processamento de transações no ponto-de-venda, controle de processo de fabricação em tempo real)."

Também segundo Pressman (2006), "[...] todo software é iniciado por alguma necessidade do negócio [...]", partindo desde simples correções em aplicações existentes até a criação de novos sistemas, produtos ou serviços.

Grandes empresas costumam ter diversos sistemas customizados para responderem as suas necessidadades de utilização, o que torna a manutenção destes extremamente cara, visto que as correções têm que ser executadas por equipes da própria empresa ou por outras empresas terceiras contratadas especificamente para isso, ao contrário dos *softwares* de prateleira, que são mantidos por suas empresas criadoras.

2.2 APLICAÇÕES WEB

Com o advento da *web*, muitos aplicativos começaram de suas plataformas *desktop* para esta nova plataforma. Inicialmente, a grande motivação para se converter um aplicativo de área de trabalho para a *web* era a facilidade de acesso a este, visto que se o servidor estivesse público na *internet*, tal aplicativo poderia ser acessado de qualquer lugar que tivesse um computador conectado à rede. Além disso, todo o processo de gerência de atualizações se tornava muito mais simples, visto que uma vez que estas estivessem publicadas no servidor, todos os clientes receberiam instantâneamente tais atualizações. Além disso, aplicações *web* são independentes de paltaforma, o que ajudou muito na sua popularização.

Pressman (2006, p. 07) define uma aplicação *web* da seguinte forma:

> "Aplicações da *Web* , 'ApsWeb', cobrem uma ampla gama de aplicações. Na sua forma mais simples, ApsWeb podem ser pouco mais que um conjunto de arquivos ligados por hipertexto que apresentam informações usando texto e poucos gráficos. No entanto, conforme as aplicações de comércio eletrônico (*e-commerce*) e B2B crescem em importância, as ApsWeb evoluem para sofisticados ambientes computacionais que fornecem não apenas caracterísiticas isoladas, funções de computação e conteúdo para o usuário final, mas também estão integradas ao banco de dados da empresa e às aplicações do negócio."

Os grandes problemas verificados em aplicações *web* são a interface com o usuário, que geralmente é mais simples e oferece menos usabilidade que uma aplicação *desktop*, a não possibilidade de trabalho estando desconectado da rede mundial de computadores ou da rede local, em caso de aplicativos utilizados em *intranets*, e a alta quantidade trocados entre o cliente e o servidor, visto que todas as informações de estado são armazenadas remotamente.

2.3 RIA (RICH INTERNET APPLICATIONS)

A indústria de software já sofreu diversas mudanças de paradigmas em um breve período de tempo. Primeiramente saiu-se do modelo de programação baseado em grandes *mainframes* com terminais burros para o modelo de cliente/servidor. Essa mudança gerou ganho de produtividade para os usuários, e os sistemas *mainframe* foram condescendentemente rotulados como ultrapassados (Fain et al., 2007, p. 02). A medida que tais aplicações começaram a ficar muito complexas, o modelo de desenvolvimento da indústria de software passou novamente a realizar grandes processamentos no servidor, trocando a experiência de utilização do usuário pela transparência na implementação e atualização dos clientes.

Para Vieriu e Tuican isso representou "[...] um retrocesso nas funcionalidades porque a maioria do processamento passou a ser feita no servidor, e a máquina cliente rodando o navegador *web* se tornou um terminal burro".

Nesse contexto, as aplicações de internet rica (RIA, do inglês *Rich Internet Applications*) surgem então como uma forma de restaurar o poder das aplicações *desktop*, entretanto com estas sendo acessadas através de páginas da *web*. RIAs rodam em máquinas virtuais (por exemplo *Adobe Flash Player*) e tem o potencial para se tornar aplicações *desktop* completas em um futuro próximo (Fain et al., 2007, p. 2). Muitas das manipulações de dados (ordenação, agrupamento e filtros) podem ser feitas localmente, como nos tempos das aplicações cliente/servidor. Analistas da indústria de *software* já prevêem que nos próximos três ou quatro anos a maioria dos novos projetos serão desenvolvidos com tecnologias RIA.

Busch e Koch (2009) definiram as características de uma aplicação RIA como sendo "a não necessidade de atualizações de página, facilidades *drag&drop*, tempo de resposta curto e animações mutimídia". As autoras complementam também essas características nos dizendo que baseado nelas, "differentes funcionalidades como validações no cliente, auto completar, atualizações periódicas e até editores de texto ricos podem ser oferecidos aos usuários".

Segundo Fain et al., (2007, p. 2), "uma aplicação de internet rica combina os benefícios de usar a Web como modelo de transferência da aplicação para o usuário com uma rica experiência de utilização que é no mínimo tão boa quanto as aplicações desktop de hoje [...]".

Como pode-se notar no trecho transcrito acima, as aplicações RIA não são baseadas em páginas com as antigas aplicações *Web*, o que facilita a gerência do armazenamento das

ações do usuário, visto que elas podem ser mantidas localmente, e não mais em sessões *HTTP* armazenadas no servidor da aplicação. Aplicações RIA eliminam a necessidade de atualizações de página pois são *stateful*, ou seja, armazenam as informações necessárias no cliente. Fain et al. (2007, p. 03) reforçam essa idéia, dizendo que "aplicações RIA não são um conjunto de páginas controladas pelo servidor, elas são verdadeiras aplicações rodando no computador do cliente é comunicando-se com os servidores basicamente para processamento e troca de dados".

Algumas diferenças arquiteturais das aplicações RIA com as antigas aplicações *Web* foram definidas por Busch e Koch (2009) como sendo a distribuição dos dados, a distribuição do controle de páginas e a comunicação do cliente com o servidor. Para as autoras, no antigo modelo *Web* os dados eram apenas armazenados no servidor, porém na aplicações RIA, esses dados passam a estar distribuídos entre o servidor e o cliente. O mesmo ocorre com o controle das páginas, visto que decisões de atualizações destas passam a ser tomadas pelo aplicativo rodando no cliente, e não mais no servidor. A última mudança arquitetural citada pelas autoras é a comunicação entre o cliente e o servidor, que em aplicações RIA pode ser tanto síncrona quanto assíncrona, em contraste as aplicações *web* assíncronas.

> A Macromedia introduziu a expressão aplicações ricas para internet em 2002, em contraste com a Web legada, hoje conhecida como Web 1.0. E logo as primeiras aplicações RIA apareceram, em 1995, quando foi criado Java. A popularidade inicial do Java manifestou-se em uma onda de pequenos programas para download chamado applets Java. Applets, criados com a biblioteca Java AWT (e mais tarde Swing) foram executados pelas Máquina Virtuais Java (JVM) dos navegadores. Ironicamente, a mesma tecnologia que fez o Java popular não foi explorada e hoje o Java brilha principalmente nos servidores e em dispositivos móveis. (Fain et al., 2007, p. 2).

2.3.1 Categorias das RIAs

Diversas categorizações podem ser encontradas na internet para as RIAs, neste trabalho nos concentraremos nas categorias definidas por Busch e Kock (2009). Para as autoras, as RIAs podem ser classificadas da seguinte forma:

- Segundo a utilização de ambiente de execução;
- Quanto à possibilidade de utilização *off-line;*
- Quanto à utilização dos recursos do cliente e do servidor;

- Quanto ao tipo de estrutura de navegação;
- Quanto à segurança.

A primeira categoria pode ser definida como as RIAs que usam exclusivamente tecnologias nativas dos navegadores, como *JavaScript*, HTML (HyperText *Markup Language*) e XML (*Extensible Markup Language*), e as que necessitam de um *plugin* instalado no navegador (por exemplo, *Adobe FlexAdobe Flex*) ou outro tipo de software instalado no cliente (por exemplo, *Java Web Start*).

A segunda categoria separa as RIAs entre as que possibilitam trabalho *off-line* e as que não tem essa funcionalidade disponível. O *Google Gears* é um exemplo de plataforma que possibilita o trabalho *off-line.*

A próxima categoria classifica as RIAs quanto ao acesso aos dispositivos do cliente e do servidor. Por exemplo, algumas RIAs podem acessar os dispositivos dos clientes para adaptar suas configurações ao *hardware* alvo.

A estrutura de navegação também é considerada um fator de classificação, contudo esta é mais uma questão de arquitura do sistema, não interferindo tanto na experiência de utilização do usuário.

Como última categoria temos as questões relacionadas a segurança do *framework*, sendo utilizados como parâmetro de classificação para esse quesito a utilização de encriptação nas comunicações com o servidor e como os dados armazenados no cliente são mantidos seguros.

2.3.2 Características de alguns *frameworks* RIA

Fazer um comparativo entreos diversos tipos de *frameworks* RIA existentes seria um tarefa extremamente difícil devido a infinidade de plataformas existentes. Devido a isso, utilizaremos nesse tópico o comparativo entre estes *frameworks* desenvolvido por Busch e Kock (2009).

No Quadro 1 são apresentados algumas plataformas *Ajax* e suas características. A grande vantagem de se utilizar estes *frameworks Ajax* é não necessidade de instalação de componetes adicionais ao *browser*, visto que estes só utilizam tecnologias nativas dos navegadores.

Nome	Linguagem de Programação	Outras Características
Prototype JavaScript Framework	*JavaScript*	Biblioteca *JavaScript.* Usado por exemplo no *Ruby on Rails.*
Script.aculo.us	*JavaScript*	Extensão do *Prototype JavaScript Framework.*
MooTools	*JavaScript*	Baseado no *Prototype JavaScript Framework.*
jQuery	*JavaScript*	Pode no futuro ser incluído no *Microsoft Visual Studio* (*ASP.net, Ajax* e *ASP.net MVC Framework*) e na plataforma de tempo de execução da *Nokia.*
Yahoo! UI Library (YUI)	*JavaScript*	Inclui diversos controles de interface, como editores de texto ricos e fornece um grande número de utilitários.
Ext JS	*JavaScript*	Originalmente uma extensão do *Yahoo! UI Library.*Fornece vários *widgets.*
Dojo	*JavaScript*	Inclui um sistema de empacotamento e *widgets.*
Google Web Toolkit	*JavaScript, Java*	Interface Nativa *JavaScript*, permite escrever partes do código *JavaScript* com o código *Java.*

Quadro 1 – *Ajax frameworks*

Fonte: Busch e Kock (2009).

Como visto nas categorias de *frameworks*, também existem plataformas que necessitam de instalações adicionais para poderem ser executadas na máquina cliente. O Quadro 2 e o Quadro 3 mostram um breve comparativo entre esses *frameworks*.

Nome	Desenvolvedora	Linguagem	Ambiente de Execução	Código Aberto	Outras Características
Java Applets	*Sun Microsystems*	*Java*	*Java Runtime Environment (jre)*	Parcialmente	Normalmente roda com o navegador. Tem acesso ao contexto do *browser.*
JavaFX	*Sun Microsystems*	*JavaFX Script*	*JavaFX Runtime* (vem com a jre)	Parcialmente	Pode ser usado no *desktop,* nos navegadores e em aparelhos móveis.
Adobe Flash	*Adobe Systems*	*ActionScript*	*Plugin Flash Player*	Apenas o *Open Source Flash.*	Focado no áudio, vídeo e animações.
Adobe Flex	*Adobe Systems*	MXML, *ActionScript*	*Plugin Flash Player*	não	Ambiente de execução *FlexBuilder* (baseado no *eclipse*)
Microsoft Silverlight	*Microsoft*	Linguagens *.Net*, linguagens de *Script* e XAML	*Plugin Silverlight*	não	Acesso *Offline*. Inclui tocador de mídia. Similar ao *Windows Presentation Foundation* (WPF)
OpenLaszlo	*Laszlo Systems*	LZX, *JavaScript*	*Flash, JavaScript*	sim	Similar ao *Adobe Flex.*

Quadro 2 – *container frameworks* parte 1

Fonte: Busch e Kock (2009).

Nome	Desenvolvedora	Linguagem	Ambiente de Execução	Código Aberto	Outras Características
UniPaaS	*Magic Software*	*UniPaaS*	*UniPaaS*	não	Permite a troca entre um cliente completo e aplicações RIA. Pode incluir componentes *.Net*
Curl	*Curl Inc.*	*Curl*	*Curl Surge* RTE	não	Container para aplicações *offline*.Utiliza o ambiente de desenvolvimento "*Curl Surge lab* IDE"
Omnis	*TigerLogic Corp*	*Omnis*	Cliente *web Omnis*	não	Ambiente de desenvolvimento integrado chamado *Omnis Studio*

Quadro 3 – *container frameworks* parte 2

Fonte: Busch e Kock (2009).

Por fim, pode-se dizer que a decisão de escolher entre um *framework* e outro dependerá muito do tipo de projeto que será construído, não se podendo dizer que uma plataforma é a melhor em detrimento das outras. Além disso, a prototipação do projeto com diversos *frameworks* pode ser tomada como último passo para a definição da plataforma que será utilizada no projeto, a fim de ser ter certeza das capacidades e deficiências de cada ferramenta.

2.4 DISCUSSÃO

Uma das motivações para o desenvolvimento deste projeto desde o princípio, como se pode notar nos objetivos específicos, era produzir um sistema que tivesse uma interface gráfica agradável ao usuário e com baixa curva de aprendizado. Partindo deste objetivo, e com base no que foi apresentado nos comparativos apresentados neste capítulo, pôde-se tomar a decisão pela utilização da plataforma *Adobe Flex* para o desenvolvimento deste sistema.

Além disso, o presente capítulo tem uma importância crucial para explicar as mudanças de paradigma que o desenvolvimento de sistemas computacionais têm sofrido nos últimos anos, e como tais mudanças influenciaram nas decisões tomadas em relação a este projeto.

Após a escolha da plataforma que iríamos utilizar para o desenvolvimento do projeto, iniciou-se o processo de definição da arquitetura do sistema, que será exposto detalhadamente no próximo capítulo.

3 ARQUITETURA DO SISTEMA

O capítulo em questão procura descrever de que modo o sistema Ulixes foi estruturado. Ele cita os requisitos que orientaram as decisões de projeto, aborda as camadas que compõem a aplicação, fala das soluções de interface adotadas, e apresenta as tecnologias selecionadas para a implementação do sistema e as razões pelas quais foram escolhidas.

3.1 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Requisitos funcionais são aqueles que descrevem o comportamento do sistema, suas ações para cada entrada, ou seja, é aquilo que descreve o que tem que ser feito pelo sistema. Já os requisitos não funcionais são aqueles que expressam como deve ser feito, em geral se relacionam com padrões de qualidade como confiabilidade, desempenho e robustez, também são muito importantes, pois definem como o *software* deverá atender suas especificações funcionais.

O processo de levantamento de requisitos foi realizado em duas etapas. A primeira consistiu de uma análise do sistema de FSM (*Field Service Management*) utilizado correntemente pela companhia cujo problema este projeto se propõe a solucionar.

A segunda se deu através da realização de uma entrevista, na qual o responsável pelo suporte em primeiro nível do sistema atualmente em uso na empresa assumiu o papel de cliente. O Apêndice A contém as perguntas e respostas da entrevista, e os subtópicos a seguir reúnem a essência do que foi apurado durante o levantamento de requisitos como um todo.

3.1.1 Requisitos Funcionais

Através de uma análise do sistema atualmente em uso pela empresa tomada por base, e também das perguntas apresentadas nas subseções anteriores, foram apurados os seguintes requisitos funcionais para o Ulixes:

- Tela de login, que receba a matrícula do funcionário e a senha registrada para ele;
- Menu da aplicação construído dinamicamente após a autenticação, conforme as permissões que o usuário possui (seu perfil de acesso);
- Conjunto de funcionalidades de administrador, permitindo o cadastramento de novos usuários e perfis;

- Armazenamento de solicitações de atendimento técnico (vulgo *tickets*) no banco de dados;
- Cada *ticket* terá um número de protocolo associado — para que possa ser vinculado a um documento em outro sistema externo, como por exemplo, uma aplicação de CRM;
- Os *tickets* deverão também ter um status, uma data de abertura, datas agendadas de início e fim de atendimento; deverão ser categorizados por nível de prioridade e pelo tipo de serviço. Além disso, o sistema deve gravar para cada *ticket* a data limite para atendimento (SLA);
- Armazenamento dos dados do cliente relacionado ao *ticket*, como nome, CPF/CNPJ, endereço completo e número de telefone para contato;
- Interface gráfica para a inserção dos novos *tickets*;
- Interface gráfica para a alteração posterior dos dados de *ticket* inseridos;
- Tela para busca de *tickets* pelo número dos mesmos, CPF/CNPJ do cliente, ou protocolo associado;
- Menu para a extração de relatórios de totais de *tickets* por período, com a opção de fazer agrupamento segundo parâmetros: status, prioridade, tipo de solicitação, empresa técnica responsável ou funcionário técnico responsável;
- Relatórios com gráficos do tipo pizza ou barra, e também com grades de dados para que se possa ver a informação em detalhes;
- *Web service* para fazer a inserção de novos *tickets*;
- *Web service* para fazer a alteração posterior dos dados de *tickets* inseridos (particularmente o status, de forma que outros sistemas externos possam mudar o status da solicitação conforme os processos de negócio da empresa);
- Serviço agendado que despache automaticamente o *ticket* para o dispositivo móvel de um técnico, cerca de uma hora antes do horário agendado para o atendimento;
- Interface gráfica para fazer o envio manual dos *tickets* para técnicos especificados pelo usuário;
- O sistema deve ainda implementar uma solução para que o técnico possa alterar o status do *ticket* utilizando para tal um dispositivo móvel.

3.1.2 Requisitos não Funcionais

Utilizando a análise prévia do problema e a entrevista apresentada, foram estimados para o projeto os requisitos não funcionais a seguir:

- Aplicação com alto grau de portabilidade, independente de sistema operacional.

3.2 ESTRUTURA DO SISTEMA

A aplicação Ulixes foi segmentada em dois projetos separados. O primeiro, que contém as telas de interface com o cliente, é um projeto em Java e Adobe Flex. O segundo é um projeto Java EE (Enterprise Edition) e agrupa os *web services*, o EJB (Enterprise Java Bean) responsável pelo disparo automático dos tickets e também algumas servlets. Optou-se por fazer tal divisão por uma questão de organização, e para não misturar componentes de tecnologias muito distintas no mesmo projeto, o que poderia resultar em conflitos de *framework* ou outros *bugs* de diagnóstico e resolução complexos.

Dentro de cada projeto, as classes estão distribuídas segundo uma interpretação flexível do padrão arquitetural MVC (*Model View Control*). As classes do tipo DAO (*Data Access Objects*) e os objetos de valores representam a camada de modelo; as diversas classes do pacote *business* reúnem a lógica da aplicação, fazendo o papel de camada de controle; e por fim, as telas MXML e a classe que expõe os *web services* fazem algumas validações básicas e renderizam a informação para o cliente (seja ele um usuário ou um processo externo), como é próprio da camada de visualização.

Detalhes a respeito das classes que formam o sistema, do projeto do banco de dados e do fluxo dos processos estão mais adiante nesse mesmo capítulo.

3.2.1 Tecnologias Utilizadas

Para o projeto Ulixes, foram escolhidas duas tecnologias principais. Para a parte de interface – conhecida como *client-side*, pois em uma aplicação web representa a parcela do *software* à qual o usuário tem acesso direto – foi eleito o *framework* Adobe Flex. O lado do servidor, ou *server-side*, foi implementado utilizando-se a plataforma Java Enterprise Edition versão 1.6. A comunicação entre ambos é feita com o auxílio de uma biblioteca de código aberto chamada BlazeDS. Cada um dos três será explicado com maior profundidade nos subtópicos que se seguem.

3.2.1.1 Adobe Flex

Conforme McCune e Subramanian (2008) resumem, "Adobe Flex é uma plataforma de desenvolvimento de aplicações que pode ser usada para construir *Rich Internet Applications* (RIAs). As aplicações Flex são baseadas na web, mas oferecem níveis imersivos de interatividade e experiências ricas de mídia, que as fazem parecer mais com programas de computador estilo *desktop* do que com as aplicações web tradicionais".

A tecnologia Adobe Flex na realidade envolve a utilização combinada de três linguagens de programação – Actionscript 3, MXML e FXG. Através de uma compilação das definições propostas por Fain et al. (2007) e Grassner (2010), pode-se entender o que é cada uma delas, e como interagem entre si.

MXML é uma linguagem de marcação (*markup*) baseada em XML, que permite a criação de componentes visuais usando programação declarativa. As *tags* adicionadas ao programa representam objetos Flex, visíveis ou invisíveis. A maior parte dos elementos em MXML correspondem a uma classe ActionScript 3, que é fornecida pela Adobe como parte da biblioteca de classes do Flex. Desenvolvedores podem criar seus próprios componentes MXML, estendendo os já disponíveis no *framework*.

Uma linguagem orientada a objetos completa, o ActionScript 3 é baseado na versão *draft* da especificação escrita para a quarta versão do ECMAScript (que serviu de ponto de partida também para o famoso Javascript). O ActionScript possui a maioria dos recursos das linguagens orientadas a objeto, como sintaxe de definição de classes, estruturação de classes em pacotes, variáveis fortemente tipificadas e mecanismos de herança.

Por fim, o FXG (*Flash XML Graphics*) é o sucessor do MXML na quarta versão do Flex, tendo herdado suas características gerais.

Independente da forma como foi escrita, quando uma aplicação Flex é compilada, todo o seu código MXML é convertido em ActionScript 3 puro. Por isso, o MXML pode ser visto como uma "linguagem de conveniência", que torna a tarefa de programar mais simples e fácil do que seria se o desenvolvedor utilizasse somente ActionScript.

Uma das principais vantagens da plataforma como um todo é que, ao empregar o Flex na implementação de um *software*, o resultado é uma aplicação para o ubíquo Flash Player – uma máquina virtual multimídia que executa arquivos em *bytecode* com a extensão SWF.

Quando um usuário acessa uma aplicação web feita em Flex, ele recebe como resposta do servidor um arquivo SWF. No momento em que o navegador processa esse retorno, o compilador JIT (*Just In Time*, ou em tempo real) do Flash Player converte o pseudo-código

para instruções nativas do computador no qual a aplicação está sendo visualizada. Isso proporciona uma performance mais rápida.

Pelo fato do Flex, em sua versão 4, ter como alvo uma única plataforma (o Flash 10), não existe preocupação com questões de compatibilidade. O sonho *WORA* (*Write Once, Run Anywhere*, ou "codifique uma vez, e execute em qualquer lugar") que os programadores Java possuíam no passado pode finalmente ser realizado através do Adobe Flex (Armstrong, 2010).

O vínculo entre a tecnologia Adobe Flex e o Flash Player torna-se particularmente atrativo ao levar-se em consideração o panorama apresentado por Fain mais de uma vez no livro "Rich Internet Applications With Adobe Flex and Java". Como o autor enfatiza, o Flash Player tem um alcance muito maior do que qualquer navegador individual ou tecnologia *client-side*, pois está instalado em cerca de 98% dos computadores *desktop* com acesso à internet. O plugin da Adobe é uma máquina virtual eficaz, cuja versatilidade multiplataforma é transparente para usuários e desenvolvedores, e as pessoas de um modo geral estão confortáveis com a idéia do Flash e não resistem à sua instalação ou atualização. Ou seja, ao ancorar-se no Flash Player, o Flex torna-se altamente portável, proporcionando ao Ulixes uma das qualidades almejadas desde a concepção do sistema.

Por fim, o último – mas significativo – fator que torna a plataforma Adobe Flex extremamente conveniente para a criação de telas e interfaces é a vasta coleção de componentes visuais customizáveis que ela disponibiliza. Estes componentes são relativamente fáceis de se utilizar e de configurar, e proporcionam um resultado muito bom tanto no aspecto estético quanto na capacidade de oferecer *feedback* ao usuário. Os elementos de geração de gráficos dinâmicos merecem especial destaque, pois são excepcionalmente bem elaborados.

3.2.1.2 Java Enterprise Edition

Sendo uma linguagem de programação que existe há pelo menos 16 anos (foi anunciada formalmente pela Sun Microsystems em uma conferência no ano de 1995), o Java dispensa apresentações. Além de oferecer todos os recursos que se espera de uma linguagem orientada a objetos completa, o Java traz como vantagem adicional a portabilidade.

Como explica a introdução feita por Deitel no livro "Java How To Program" (2004), essa qualidade é atingida através da utilização de pseudo-código. O compilador Java converte o código em alto nível escrito pelo programador para *bytecodes*, que representam as tarefas a serem realizadas durante a execução do programa; estes *bytecodes* são então interpretados

pela JVM, a *Java Virtual Machine*. Um aplicativo de *software* cujo objetivo é o de simular um computador, a JVM oculta o sistema operacional e o *hardware* subjacentes dos programas que interagem com ela. Dessa forma, os *bytecodes* passam a ser instruções independentes de plataforma, que podem executar em qualquer computador contendo uma JVM capaz de reconhecer a versão do Java na qual o pseudo-código foi compilado.

Além desse aspecto positivo referente à versatilidade, a escolha do Java para a implementação do *server-side* do sistema Ulixes deveu-se, também, a algumas facilidades oferecidas pela plataforma Java Enterprise Edition 1.6. A tarefa de exportar os métodos das classes de negócios na forma de *web services* foi extremamente simplificada pelo uso das chamadas *annotations*. Empregando-se em conjunto o Java EE 6 e o servidor de aplicações Glassfish, basta acrescentar a anotação "@WebService" sobre a declaração da classe cujos métodos se deseja expor na forma de serviços, e a exportação das operações e a geração do contrato WSDL são feitos de modo transparente para o desenvolvedor.

Outro recurso interessante da plataforma Java EE que teve importância fundamental para o Ulixes foi o assim chamado *Timer Service* (informações detalhadas sobre o mesmo podem ser encontradas no "Java EE 6 Tutorial", disponibilizado pela Oracle). O *Timer Service* é um *stateless Enterprise Java Bean*, ou seja, uma classe Java que pode ser invocada remotamente e que não preserva alterações ou dados de uma chamada para a seguinte – e por isso é classificada como *stateless*. Mais importante, no entanto, o *Timer Service* (que poderia ser livremente traduzido como "Serviço de Temporizador") pode ser configurado de maneira que o servidor o ative automaticamente em certos dias e horários pré-determinados. Esta característica do *Timer Service* permitiu a idealização e implementação de um processo de envio de tickets periódico, que não precisa da intervenção de nenhum usuário para disparar sua execução.

3.2.1.3 BlazeDS

Como Tiwari sumariza muito bem em seu livro "Professional BlazeDS: Creating Rich Internet Applications", o Java tranquilamente ocupa uma posição entre as mais populares plataformas de programação. OAdobe Flex, por sua vez, é um proeminente *framework* de desenvolvimento de *Rich Internet Applications*. Assim, a combinação de ambos torna-se uma proposta das mais atraentes para as empresas que desejam criar aplicações envolventes e ricas, seja para aumentar a produtividade ou a satisfação dos clientes.

Uma integração efetiva do Java com o Flex implica a possibilidade de construir aplicações robustas e com escalabilidade do lado do servidor, e ao mesmo tempo desfrutar das capacidades superiores de renderização do Flash Player. Essa comunicação entre as linguagens pode ser feita através de diversos produtos e ferramentas, incluindo o *LifeCycle Data Services* (LCDS) e o BlazeDS, ambos da Adobe. O LCDS é um produto comercial, e o BlazeDS é a alternativa de código aberto ao LCDS.

Conforme Kore (2009) define, BlazeDS é uma tecnologia de acesso remoto e mensageria, utilizada em servidores. Sendo de acesso remoto, ela simplifica a comunicação entre o Flex e o Java, serializando e de-serializando os objetos que transitam entre ambos. Em termos mais simples, ao usar o BlazeDS, os desenvolvedores podem chamar classes e métodos Java existentes no servidor, de dentro da própria aplicação Flex. A comunicação é assíncrona: o Flex invoca o BlazeDS, registrando um método de *callback*, e o BlazeDS aciona a classe Java desejada; quando obtiver o retorno da chamada, ele o encapsula em um objeto próprio e entrega esse objeto ao método registrado pelo Flex.

O BlazeDS em si é implementado em código Java, e funciona dentro de uma *Java virtual machine* (JVM). Mais precisamente, é um pequeno aplicativo web que pode ser publicado em um servidor de aplicações Java ou mesmo em um contâiner minimalista de Servlets, como o Apache Tomcat. Estas características o tornam altamente portável.

Decidiu-se empregar o BlazeDS como solução de integração do Ulixes por duas razões: por ser uma biblioteca de código aberto, e porque sua configuração não é uma tarefa de grande complexidade. Infelizmente, para sistemas de grande porte, o BlazeDS não apresenta um desempenho tão satisfatório quanto o LCDS. Contudo, a licença comercial do LCDS é por demasiado custosa, e estava fora das possibilidades financeiras deste projeto.

Como ambas as ferramentas de integração são da Adobe e guardam alguma compatibilidade entre si, o uso do BlazeDS em um primeiro momento não chega a ser um problema. O sistema Ulixes poderia ser facilmente migrado para o LCDS em caso de necessidade, bastando para isso trocar a biblioteca do BlazeDS pela do *LifeCycle Data Services*, sem qualquer alteração adicional no código-fonte.

3.3 DIAGRAMAS DE CASOS DE USO

Uma vez esboçada a estrutura da aplicação Ulixes – para uma melhor compreensão da mesma em nível macro – os diagramas que se seguem dedicam-se a descrever em profundidade o sistema projetado.

3.3.1 Diagrama de Casos de Uso Administrador

O diagrama da Figura 1 representa os casos de uso referentes à gerência do sistema de solicitações de tickets, ou seja, aborda casos de uso que estarão disponíveis com usuários com perfil de administrador do sistema.

Neste contexto estão inseridos os casos de uso relativos ao cadastro e modificação de empreiteiras, técnicos, perfis e usuários. É importante destacar que tais casos de uso só devem ser manipulados por pessoas com conhecimento prévio sobre o processo de atendimento de chamadas, visto que dados erroneamente cadastrados podem influenciar na forma como o sistema irá despachar os tickets.

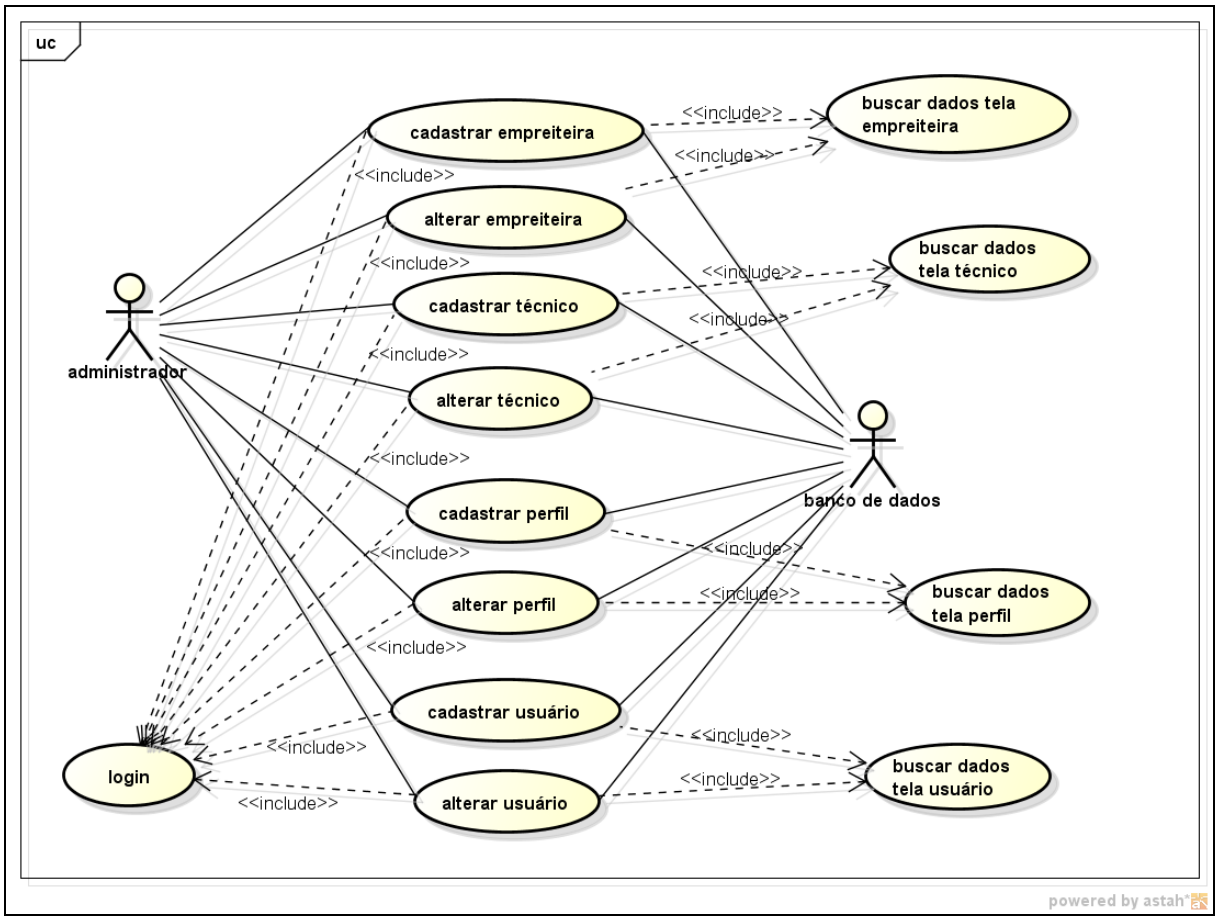


Figura 1 – Diagrama de Casos de Uso Administrador

Fonte: Autoria própria.

3.3.2 Diagramas de Casos de Uso Gerais

O diagrama da Figura 2 aborda os casos de uso referentes não à manutenção da aplicação, mas às ferramentas que o Ulixes oferece para manipulação de solicitações de atendimento (tickets) e de cadastro de clientes.

Para os casos “Inserir Ticket”, “Alterar Ticket” e “Consultar Cliente”, existe uma segunda versão de caso de uso que representa a mesma funcionalidade com interface diferenciada. Ao invés de disponibilizar uma tela para o usuário, estes casos foram implementados na forma de *web services*, permitindo a processos externos – como sistemas de CRM – conectarem-se ao sistema do Ulixes e interagirem com o mesmo.

O caso de uso “Despachar Ticket” destaca-se dentre os demais, pois o ator externo que inicia o processo não é um usuário ou uma aplicação, e sim um *timer* – a classe que despacha os tickets é um EJB (*Enterprise Java Beans*) ativado pelo servidor de aplicação nos horários pré-agendados.

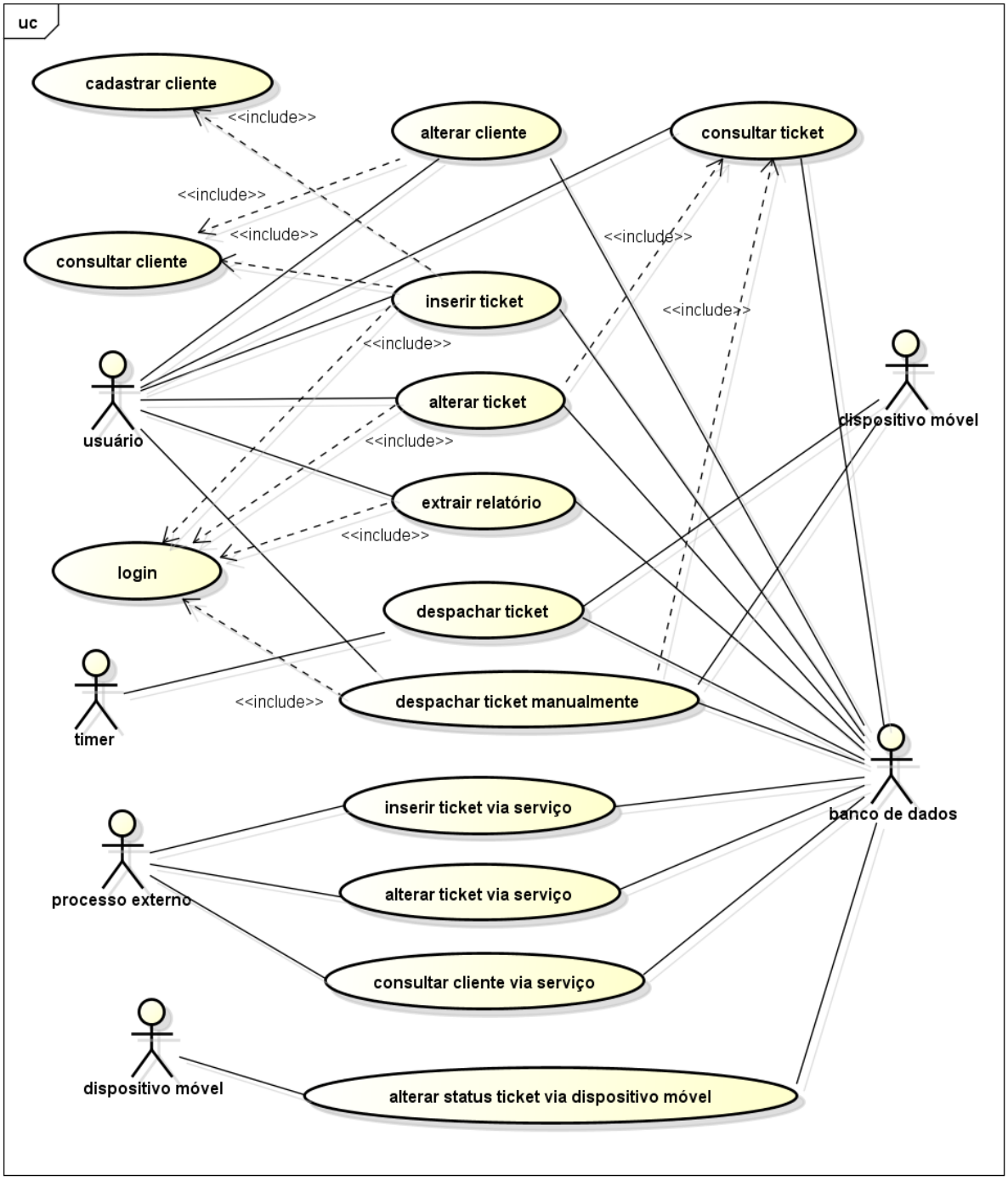


Figura 2 – Diagrama de Casos de Uso Usuário

Fonte: Autoria própria.

3.4 DIAGRAMAS DE CLASSES

Diagramas de Classes são representações de estrutura e dos relacionamentos entre as classes que servem de modelos para objetos. Esta modelagem serve de base para a construção de diagramas de seqüência, de comunicação e de estados, pois define todas as classes que o sistema precisa possuir para o seu bom funcionamento. Nessa representação, ficam conhecidos todos os atributos e métodos existentes nas classes e todos os relacionamentos e associações existentes entre tais classes. A seguir têm-se os diagramas de classes do sistema Ulixes, sendo que as classes constantes nesses diagramas foram obtidas através de levantamento realizado nos diagramas de casos de uso. Além disso, já são especificados os métodos e atributos que integrarão cada classe do projeto.

3.4.1 Diagrama de Classes Módulo Empreiteira

Como se pode observar na figura 3, o diagrama de classes do módulo empreiteira tem seu núcleo na classe Ulixes (algo que se repetirá em todos os diagramas de classes da aplicação, exceto no que representa as classes do dispositivo móvel), que ativa as classes TelaAddEmpreiteira e TelaAddTecnico.

As classes de interface com o usuário (TelaAddEmpreiteira, TelaAddTecnico e PopupNovaEmpreiteira) foram criadas com a utilização da linguagem de programação *actionscript* da *Adobe*, o que proporciona um bom aspecto visual para a interface com o usuário e uma boa integração com as classes de negócio codificadas na linguagem de programação Java.

Importantes aspectos do modelo de dados também podem ser obtido através desse diagrama de classes, visto que as relações entre as classes de transferência podem ser inferidas deste diagrama.

Outro aspecto importante que se deve observar é que esse diagrama de classes é extremamente enxuto e simples, o que de fato era um dos objetivos do sistema, facilitando futuras modificações no modelo, se estas forem necessárias.

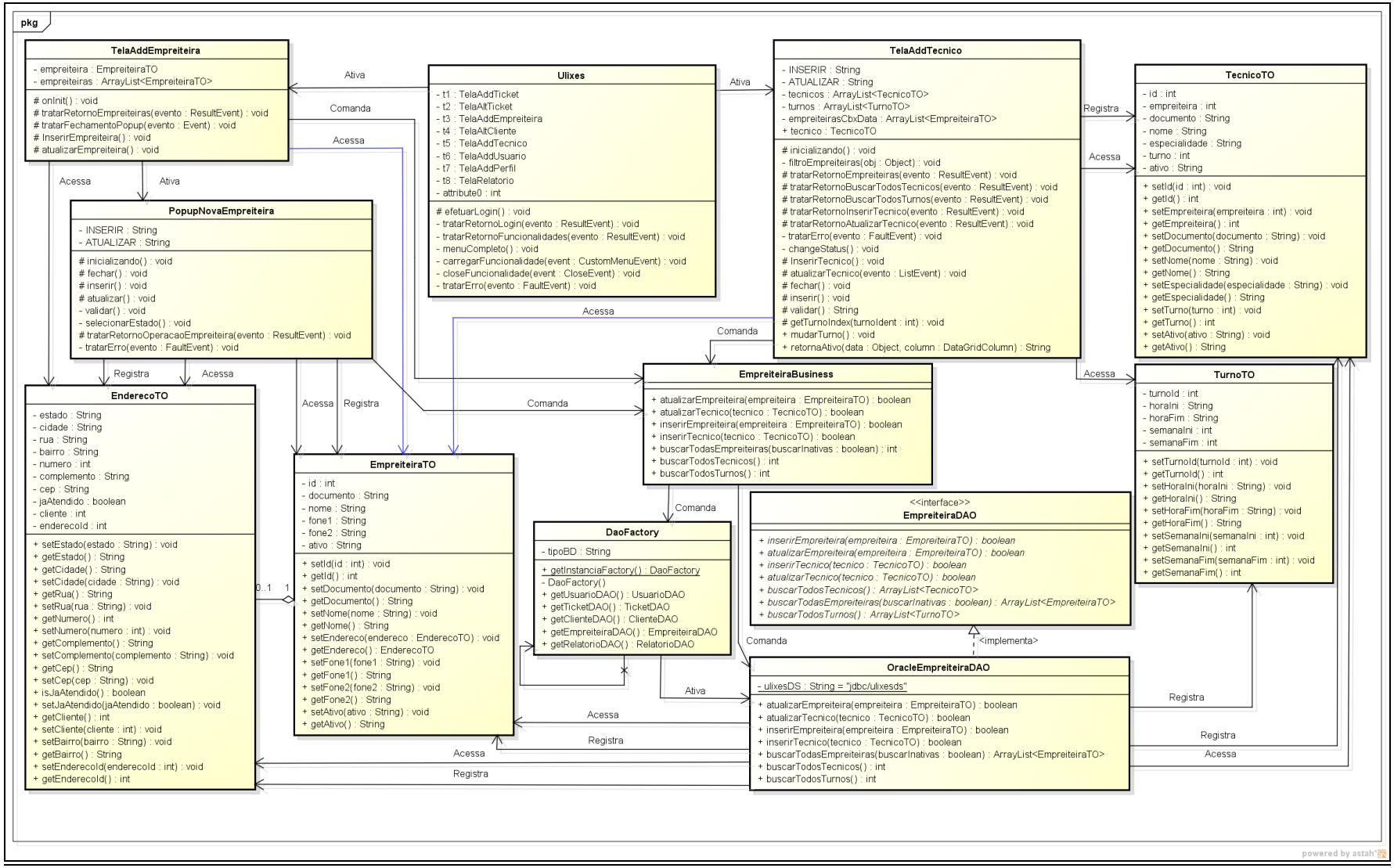


Figura 3 – Diagrama de Classes Módulo Empreiteira

Fonte: Autoria própria.

3.4.2 Diagrama de Classes Módulo Usuário

O diagrama de classes do módulo usuário (Figura 4) tem como função descrever de forma ilustrada todas as classes, relacionamentos e associações necessários para a realização dos casos de uso relativos ao cadastro e alteração de usuários e ao cadastro e alteração de perfis.

Como no diagrama de classes do módulo empreiteira, todas as classes de interface são codificadas em *actinscript*, utilizando basicamente os mesmos componentes visuais de tal diagrama, o que faz com que a curva de aprendizagem para a utilização do sistema seja realmente baixa, visto que tudo o que é aprendido sobre a interface de um determinado módulo pode ser transposto para o outro sem maiores problemas.

As classes utilizadas para a transferência de valores entre as diversas camadas do sistema, UsuarioTO, FuncionalidadeTO, PerfilTO e ItemMenu, são as responsáveispela maioria dos relacionamentos descritos nesse diagrama de classes, o que é perfeitamente justificável, visto que a função dessas classes é organizar o fluxo das informações que trafegam pelo sistema, o que exige um grande número de operações de acesso e registro.

Seguindo o padrão definido para o sistema, a classe UsuarioBusiness não acessa diretamente as classes de interface com o banco de dados, ao invés disso o acesso é efetuado através de uma interface, o que facilita a migração para outros sistemas de gerenciamento de banco de dados.

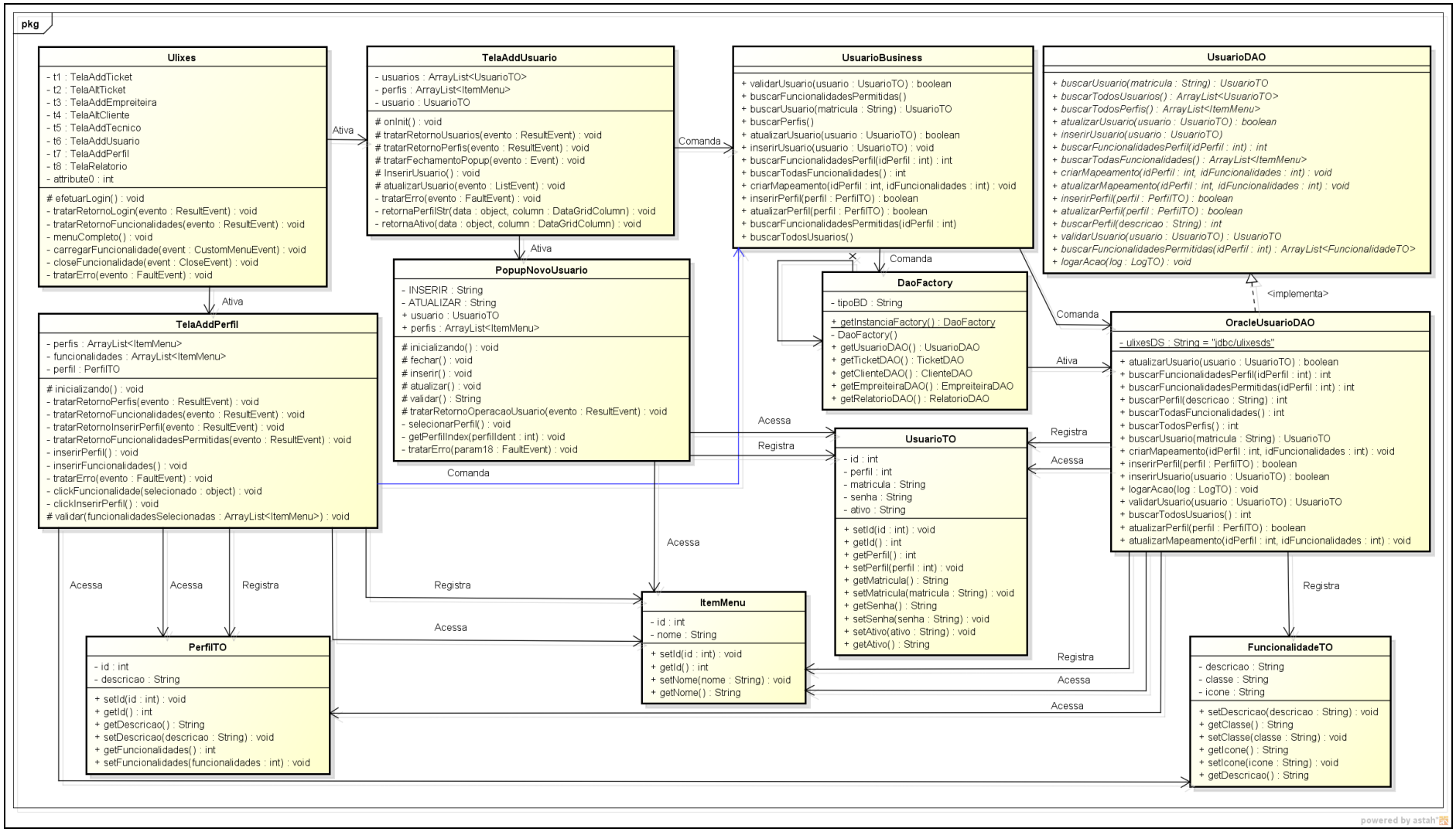


Figura 4 – Diagrama de Classes Módulo Usuário

Fonte: Autoria própria.

3.4.3 Diagrama de Classes Módulo Dispositivo Móvel

O Diagrama de Classes do Módulo Dispositivo Móvel é, sem dúvid,a o menor diagrama de classes do sistema, o que é perfeitamente compreensível se consideramos que o problema que precisa ser resolvido pelo diagrama de caso de uso que originou tal diagrama de classes é realmente pequeno.

Na Figura 5 pode-se observar que uma vez instanciada a classe UlixesApp, esta ativa a classe UlixesMainActivity, que por sua vez pode ativar todas as classes de interface com o usuário disponíveis na aplicação móvel.

A classe SmsService, única não relacionada anteriormente, realiza o serviço de verificação das mensagens da fila *SMS* (do inglês *short message service*, serviço de mensagens curtas). Esta classe não tem interface com o usuário, roda de forma independente das demais classes é interage apenas com funções do próprio sistema operacional *Android*.

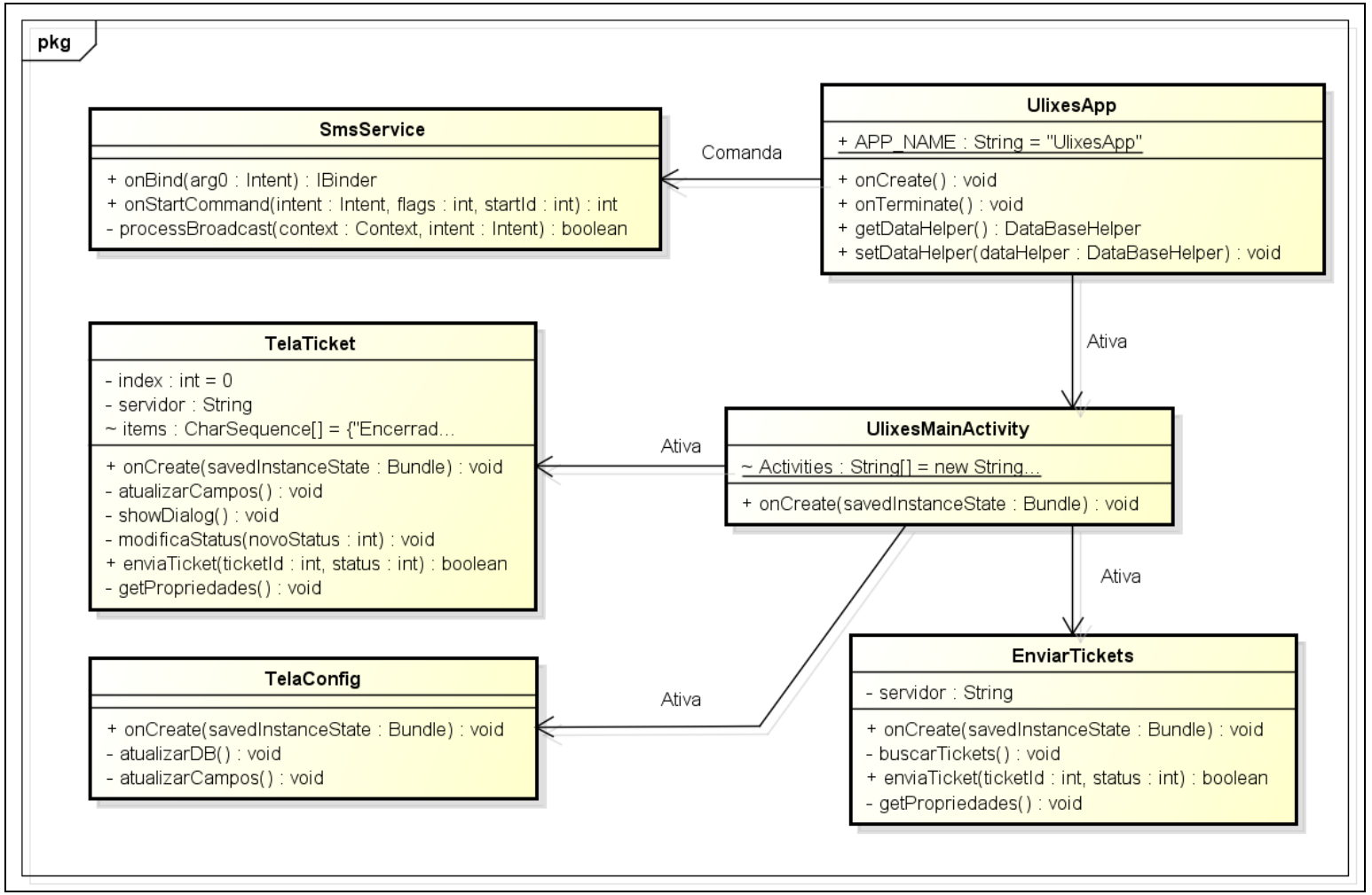


Figura 5 – Diagrama de Classes Módulo Dispositivo Móvel

Fonte: Autoria própria.

3.4.4 Diagrama de Classes Módulo Ticket

Na Figura 6 estão reunidas as classes do módulo de ticket do sistema Ulixes – o mais extenso dos diagramas.

Conforme mostra o desenho, o ponto de entrada do sistema é a classe Ulixes, que ativa todas as telas. Objetos das classes TelaAddTicket, TelaAltTicket, TelaDispTicket, TelaAltCliente, e TelaRelatorio são instanciados e disponibilizados para o usuário como conseqüência de eventos disparados no menu, que por sua vez é um componente gráfico da classe Ulixes. As classes TelaCliente e TelaEndereco estão contidas nas demais telas, permitindo o reaproveitamento de código e a uniformidade de *layout* das interfaces, ao isolar operações genéricas relacionadas aos clientes e endereços cadastrados (pode-se citar como exemplos a busca de um cliente por cpf/cnpj, e também a busca de cidade, bairro e rua por um dado CEP).

Todas as classes de interface com o usuário, feitas em actionscript e utilizando elementos do *framework* Adobe Flex, comunicam-se com o *backend* escrito em Java através de classes de negócio. As classes de negócio utilizadas no módulo são:

- TicketBusiness, empregada na maioria das operações referentes a solicitações de serviço, e também para o carregamento de listas tipo “combo” com as opções de tipo, subtipo, detalhe, status e prioridade de ticket;
- EmpreiteiraBusiness, que fornece os as informações de empreiteira, técnicos e turnos de trabalho necessárias ao envio dos tickets para campo;
- RelatorioBusiness, que encapsula os métodos responsáveis por coletar no banco os dados que alimentam os relatórios do Ulixes.

Como foi utilizado na concepção do sistema o *design pattern* conhecido como *Factory* – conforme está explicado no tópico que trata a arquitetura da aplicação Ulixes – a camada de negócios não acessa diretamente as classes que têm contato com a base de dados. Ao invés disso, cada uma das três classes citadas na lista acima utiliza a DaoFactory para obter uma interface Java, que representa o *data access object* necessitado. Através destas interfaces, as operações de banco de dados são invocadas de modo transparente.

Em um primeiro momento, optou-se por implementar as interfaces visando um banco de dados Oracle. O resultado foram as classes OracleTicketDAO, OracleUsuarioDAO, OracleRelatorioDAO, OracleEmpreiteiraDAO e OracleClienteDAO. Caso fosse solicitada pelo cliente uma versão do sistema compatível com um banco de dados PostgreSQL, para

citar um exemplo, bastaria implementar as interfaces Java de modo apropriado ao SGBD selecionado, e as outras camadas do *software* não precisariam sofrer qualquer mudança para que o novo banco funcionasse.

Por fim, no diagrama a seguir estão todos os objetos usados para agrupar valores em entidades lógicas. As classes TicketTO, AgendaTO, CategoriaTO, PrioridadeTO, UsuarioTO, LogTO, EmpreiteiraTO, TecnicoTO, TurnoTO, EnderecoTO, ClienteTO, RelatorioTO e RelatDetalheTO servem para organizar melhor as informações que transitam pelo sistema, mantendo-as unificadas de acordo com o que simbolizam; elas também simplificam as assinaturas dos métodos, que recebem estes objetos compostos ao invés de uma extensa relação de parâmetros. Vale ressaltar que algumas destas classes refletem a estrutura da base de dados e seus relacionamentos: um ClienteTO pode conter em si a referência para diversos EnderecoTO, e assim por diante.

A classe ItemMenu é um caso especial; ela serve para vincular o id de um registro com sua descrição. Quando é apresentado ao usuário um menu com as prioridades de um ticket, o provedor de dados que alimenta o menu é uma lista de objetos ItemMenu. Em cada um deles está um rótulo compreensível para o usuário, e também o valor de chave primária que corresponde àquele rótulo. Esta estrutura é conveniente quando se está elaborando uma interface gráfica Adobe Flex, pois a lista pode ser criada inteiramente com o uso de Java e, com a configuração de uns poucos parâmetros mxml, interpretada e exibida adequadamente pelos componentes prontos existentes no *framework* Flex.

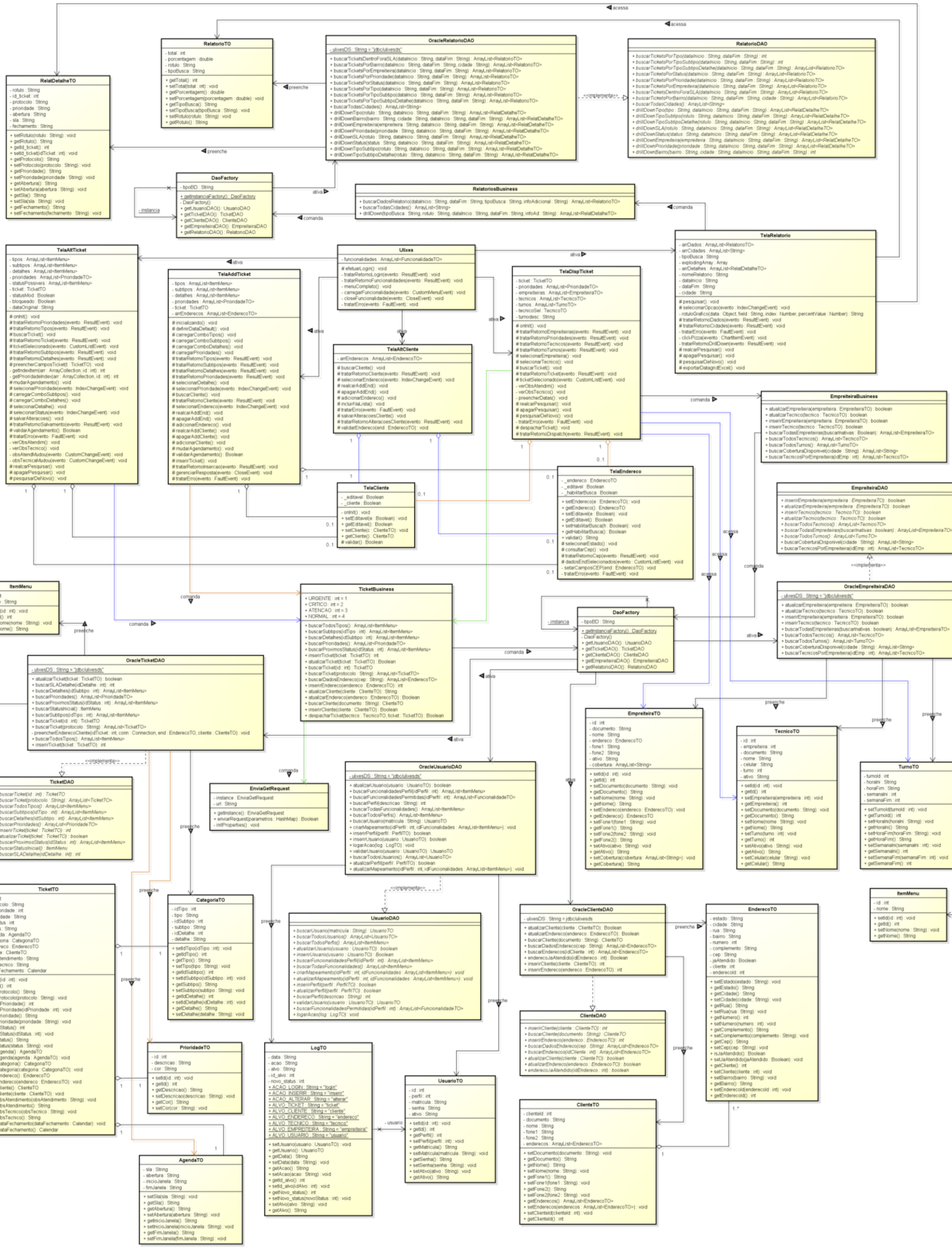


Figura 6 – Diagrama de Classes Módulo Ticket

Fonte: Autoria própria.

3.5 MODELO ENTIDADE RELACIONAMENTO (MER)

A estrutura de banco de dados do Ulixes foi concebida a partir de alguns pressupostos.

Primeiramente, desejava-se solucionar os problemas que foram detectados na aplicação de *Field Service Management* estudada como base. Um deles era a centralização da informação em poucas tabelas, cada uma delas extensa e com muitas colunas. Embora o processo de consulta seja otimizado ao se eliminarem junções (as operações SQL conhecidas como *joins*), a concentração dos dados nessas entidades causava uma sobrecarga – uma vez que praticamente toda consulta ou alteração feita no banco passava obrigatoriamente pelas mesmas duas ou três tabelas.

Assim, assumindo-se o ônus de ter consultas um pouco mais lentas, decidiu-se desmembrar as entidades. O resultado foi a possibilidade de buscar o cliente e o endereço referentes a um ticket, ou o ténico responsável por uma determinada solicitação de serviço, sem a necessidade de envolver toda a entidade Ticket nessa operação. Sendo o repositório de todas as solicitações de serviço do sistema, a tabela Ticket naturalmente será demandada mais do que as outras, e quaisquer medidas que reduzam a quantidade de acessos à mesma podem se revelar providenciais a longo prazo.

Os relacionamentos entre os elementos que compõem a aplicação Ulixes estão bastante claros na Figura 7. Todo ticket possui uma categoria, ou seja, está classificado por tipo, subtipo e detalhe. Sendo o detalhe o grau mais específico de classificação, é ele quem determina o SLA (*Service Level Agreement*) da solicitação; a empresa usuária do Ulixes pode, portanto, definir que um defeito em um *driver* de dispositivo deve ser atendido com mais presteza do que um defeito em um outro *software* qualquer – para citar um exemplo.

Ao definir a mecânica de SLA, existe um outro fator a ser considerado. Um cliente que está com um problema em um servidor de sua empresa certamente tem mais urgência no atendimento do que um usuário doméstico com uma falha em seu computador pessoal. Para refletir isso, criou-se a entidade Prioridade. Definida pelo operador no momento do cadastro do ticket, a prioridade atua como um multiplicador fracionário, que reduz a SLA originalmente prevista para a solicitação de serviço.

Um ticket, ao longo de sua história dentro do sistema, assume uma série de status distintos. Para flexibilizar o Ulixes, este fluxo não é completamente rígido; ele está definido na entidade Fluxo_Status, que determina, a partir de um dado status, todos os outros que o ticket pode assumir na seqüência. Partindo de um status inicial – “Aberto” – a solicitação de serviço pode seguir quaisquer caminhos mapeados na tabela Fluxo_Status, até chegar a um

dos extremos finais da cadeia – os estados “Encerrado” ou “Cancelado”. O despachamento de um ticket sempre ocorre quando este se encontra “Agendado” ou “Reagendado”, mas com exceção destes pontos de controle mencionados, o restante do fluxo pode ser remodelado com algumas poucas alterações no conteúdo da base de dados, o que abre algumas opções interessantes de customização do Ulixes.

Um ticket está intrinsecamente ligado a um certo endereço desde o momento de sua inclusão no sistema. O endereço, por sua vez, pertence a um cliente – e pode ser o único que ele possui, ou somente um de muitos, no caso de uma companhia com diversas filiais ou de uma pessoa física com várias residências.

Ao ser despachado, o ticket também passa a ser vinculado a um determinado técnico. O bairro e cidade no qual o endereço de atendimento estão situados determinam a empreiteira encarregada de cumprir aquela solicitação. Cada empreiteira atua em seu próprio município, em um ou mais bairros (regiões) que compõem a sua área de cobertura. Uma vez que a empreiteira responsável é identificada, o sistema faz a seleção de um técnico apropriado (ou seja, de um técnico cujo turno seja compatível com a janela de atendimento do ticket) quando o processo massivo de envio de tickets está em execução.

As entidades Usuário, Perfil, Mapeamento e Funcionalidades armazenam os dados usados na autenticação de um usuário e na geração do menu dinâmico. Um ou mais indivíduos podem compartilhar um perfil, que se traduz em um conjunto de funcionalidades de acesso permitido. Não há limites para a quantidade de perfis ativos.

Por fim, o *log* do sistema é feito na entidade de mesmo nome. Consiste em uma ação – como, por exemplo, “incluir” ou “alterar” – e em um alvo – que pode ser um ticket, um cliente, ou um outro elemento do sistema. O usuário responsável pela operação, a data da mesma e o id do registro são armazenados. No caso específico de alteração de status de ticket, a coluna novo_status também é preenchida. Esta solução torna possível usar apenas uma entidade para guardar *logs* diversificados.

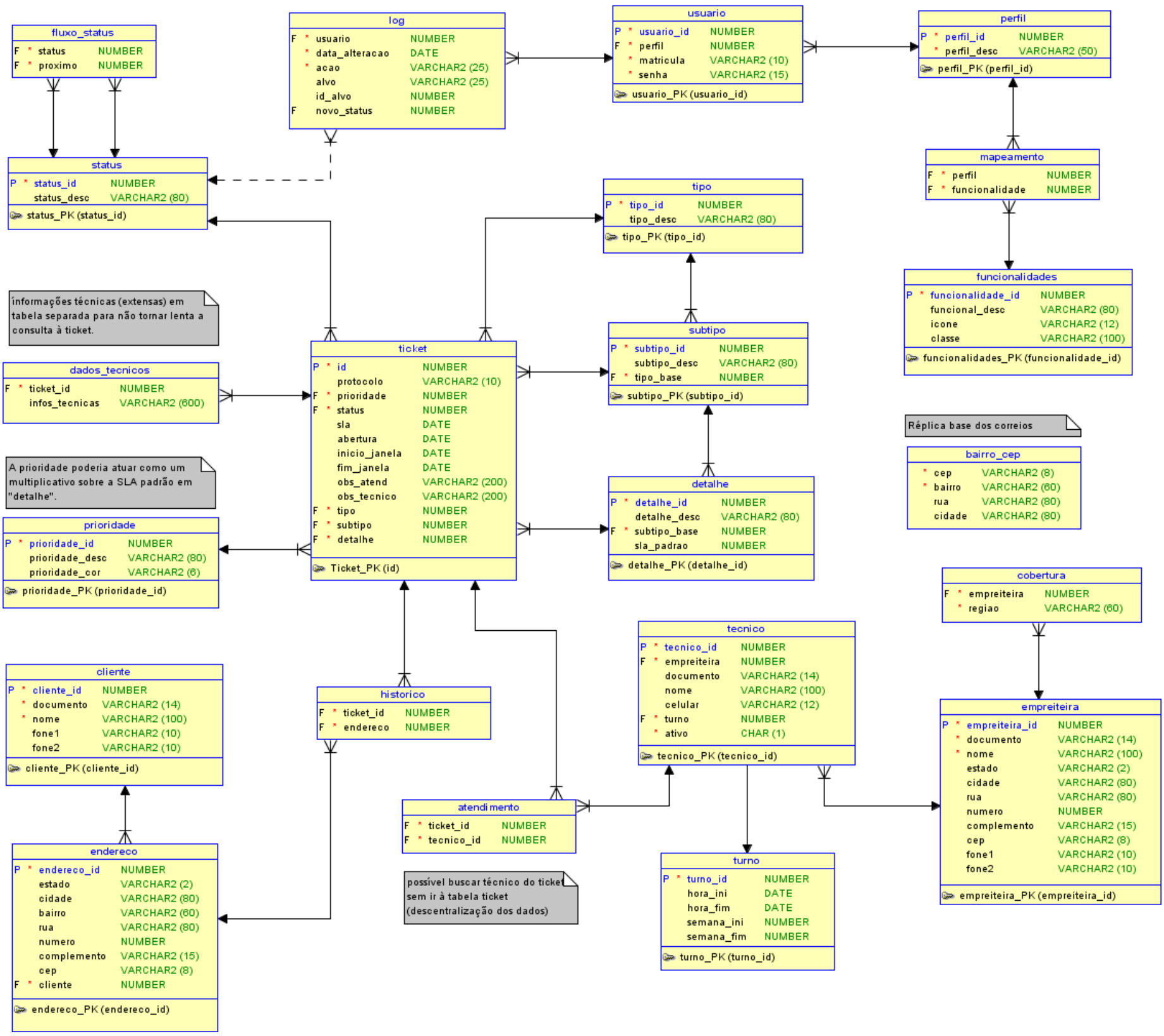


Figura 7 – Diagrama Entidade Relacionamento

Fonte: Autoria própria.

3.6 DIAGRAMAS DE SEQÜÊNCIA

As diversas funcionalidades que compõem a aplicação Ulixes estão descritas com maior detalhamento nos diagramas de seqüência. O diagrama de seqüência do caso de uso “Inserir Ticket” foi selecionado como um dos mais representativos do *software*, e por isso ele é apresentado no item 3.6.1 (logo a seguir).

Os demais diagramas, juntamente com um texto explicando o algoritmo utilizado para cada funcionalidade, estão no Apêndice B. Os primeiros subtópicos do apêndice referem-se aos casos de uso administrativos, que permitem a manutenção e atualização do sistema – através do cadastro de perfis, novos usuários, empreiteiras e técnicos adicionais. Em seguida, estão os casos de uso gerais, que tratam da manipulação de solicitações de serviço (tickets) e dados diretamente relacionados aos mesmos.

É importante fazer um comentário a respeito da comunicação entre a interface e as classes de negócio. Sendo um projeto que combina *frontend* Adobe Flex com *backend* Java, as telas são componentes MXML (*Macromedia eXtensible Markup Language*) e a camada de negócio consiste em classes Java. Como o meio de campo entre ambas as linguagens é feita pela biblioteca de uso gratuito BlazeDS, as setas que representam as chamadas que o Flex faz ao Java e vice-versa aparecem sempre como mensagens assíncronas nos diagramas de seqüência. Isso se deve à própria implementação do BlazeDS, que ao acionar o Java registra um método de *callback* na interface Flex. Quando houver um retorno do método Java invocado, o BlazeDS se encarrega de chamar a função correspondente na interface, qualificando (como já foi citado) uma comunicação assíncrona entre as camadas.

3.6.1 Diagrama de Seqüência Inserir Ticket

A Figura 8 e a Figura 9 tratam de um dos principais casos de uso da aplicação, aquele que aborda o cadastro de solicitações de serviço.

Primeiramente, o usuário é solicitado a fornecer os dados básicos do ticket. Ele deve preencher o protocolo, escolher uma prioridade no menu *drop-down* apresentado, utilizar os outros menus para definir a categoria do ticket (o conjunto de atributos que compreende tipo, subtipo e detalhe), assinalar uma data e hora para o atendimento – o início da janela de tempo na qual o técnico estará disponível para solucionar o problema – e, se desejar, acrescentar alguma observação.

Em seguida, ele tem as opções de: consultar um cliente já existente, através do cpf/cnpj do mesmo, e associá-lo ao ticket; ou cadastrar os dados de um cliente ainda não atendido, para que ele seja inserido na base juntamente com o novo ticket. Estas duas operações estão descritas em detalhe nos diagramas de seqüência respectivos (ver apêndice B).

Definido o cliente (e também o endereço vinculado à solicitação), são feitas validações para garantir que todas as informações essenciais tenham sido preenchidas. Em caso de sucesso ao validar os dados, o ticket é gravado na base de dados. A necessidade de inserir também um novo cliente e um novo endereço são avaliadas e atendidas dentro da classe de negócios, sem que o usuário precise provocar qualquer evento adicional na interface.

Por fim, uma mensagem com o identificador (id) do novo ticket é exibida se a inserção for completada sem erros.

Figura 8 - Parte 1: Diagrama de Seqüência Inserir Ticket

Fonte: Autoria própria.

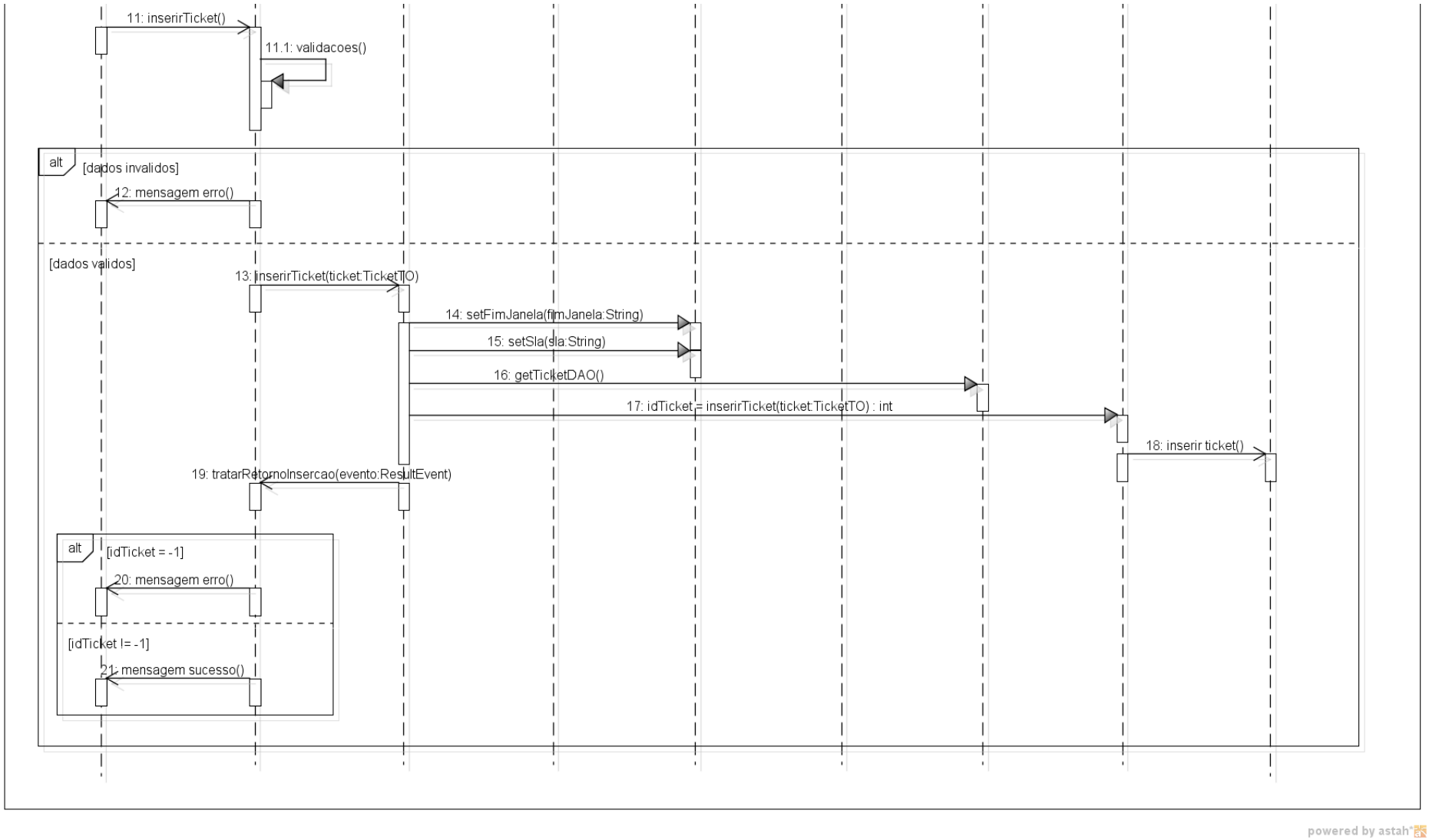


Figura 9 - Parte 2: Diagrama de Seqüência Inserir Ticket

Fonte: Autoria própria.

3.7 SOLUÇÕES DESENVOLVIDAS

Tendo apresentado as tecnologias que foram essenciais para a concretização do sistema Ulixes, pode-se agora abordar as soluções julgadas de maior interesse para o atendimento dos requisitos levantados anteriormente – os pontos de destaque do projeto.

3.7.1 Menu Dinâmico

Para permitir a configuração dos mais variados perfis de acesso, e também a fácil integração de novas funcionalidades ao Ulixes, foi implementado no *software* o conceito de menu dinâmico. Ao fazer o login na aplicação (vide Figura 10), o algoritmo não apenas valida a matrícula e a senha fornecidas, como também consulta o perfil correspondente ao usuário que está sendo autenticado.

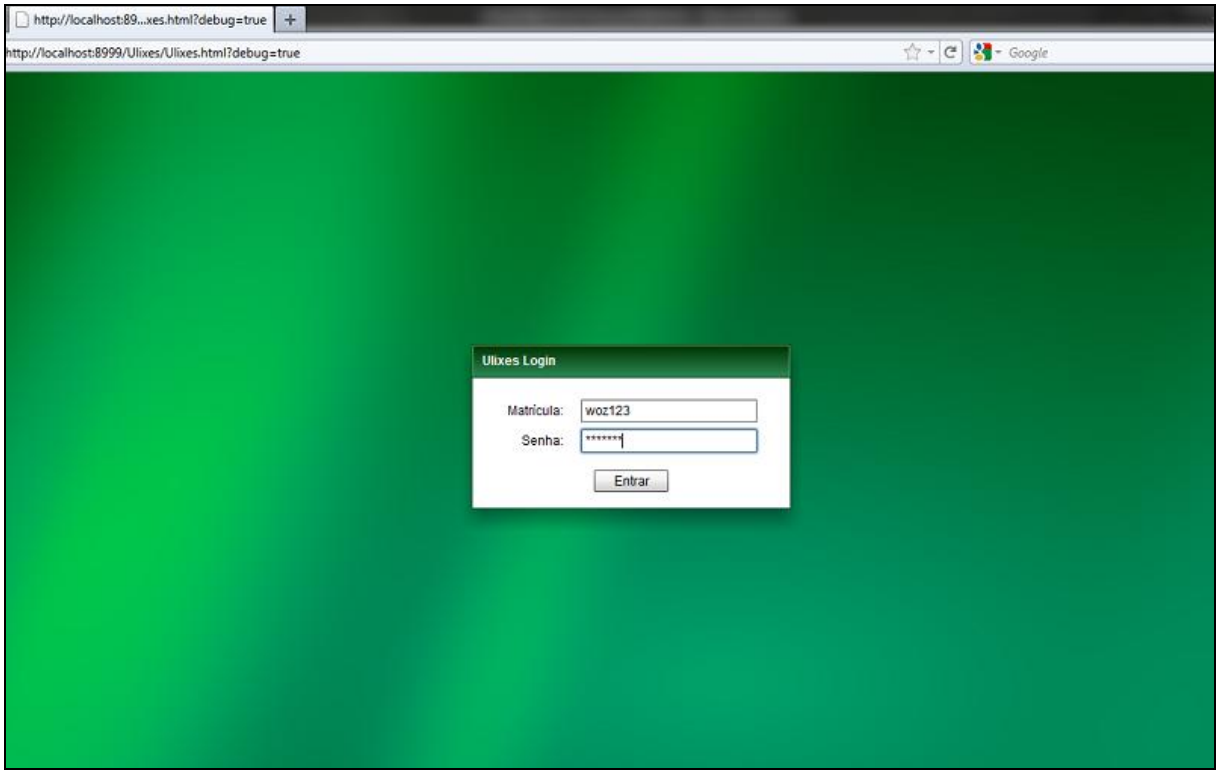


Figura 10 – Tela de Login do Sistema Ulixes
Fonte: Autoria própria.

Todas as ferramentas existentes dentro do sistema Ulixes estão cadastradas em uma tabela de funcionalidades. Entre as informações que constam na base de dados, incluem-se:

- O nome “oficial” da ferramenta (ou seja, o rótulo com o qual ela será apresentada ao usuário);
- Um caminho de arquivo, que aponta a localização de uma imagem PNG dentro do projeto (o ícone que identifica a ferramenta no menu);
- O nome completo da classe MXML que serve como tela de entrada para a funcionalidade.

Cada perfil de acesso está associado a um ou mais registros da tabela de funcionalidades; quando o perfil do usuário é identificado, a lista de telas às quais ele tem acesso autorizado é obtida do banco, e transportada para a aplicação. Vários botões, um para cada elemento da lista, são criados e inseridos no menu lateral; os títulos destes e as imagens que os adornam são definidos com base nos dados armazenados.

O detalhe mais importante deste processo é que cada um dos botões do menu mantém, em um atributo interno, o nome da tela MXML da funcionalidade à qual se refere. Quando um evento de clique é disparado a partir do menu, o botão clicado envia o valor deste atributo juntamente com as demais informações do evento. Através do nome qualificado da classe, é feita a instanciação da tela – utilizando-se, para isso, os mecanismos de reflexão disponibilizados pelo ActionScript. Isso significa que a aplicação é capaz de criar um objeto de uma determinada classe a partir de uma decisão tomada em tempo de execução, sem conhecer previamente qual a classe que será instanciada.

Essa estrutura, além de assegurar que nenhum usuário tenha acesso a telas às quais não está autorizado, também proporciona escalabilidade ao Ulixes. Para acrescentar uma nova ferramenta ao conjunto das já implementadas, basta incluir na tabela de funcionalidades certos dados básicos, e o sistema estará pronto a reconhecer o recurso recém-adicionado.

A Figura 11 ilustra o menu dinâmico com todas as funcionalidades criadas para o projeto. Detalhes mais aprofundados do processo de login estão no tópico referente ao Diagrama de Seqüência Login; uma visão geral da base de dados utilizada pelo Ulixes encontra-se junto ao Modelo Entidade-Relacionamento disponível neste documento.

Figura 11 – Menu Dinamicamente Construído, com Ícones e Rótulos Obtidos da Base de Dados
Fonte: Autoria própria.

3.7.2 Interface Gráfica

Uma das preocupações durante a elaboração das interfaces gráficas do sistema Ulixes foi a consistência das telas, e também a uniformidade visual da aplicação como um todo. Para isso, empregou-se um conjunto bem definido de cores, e usou-se em todas as funcionalidades uma estrutura padrão para a apresentação da informação.

Por uma questão de praticidade, uma ou mais telas podem ser abertas simultaneamente dentro da aplicação, ficando dispostas em abas que podem ser acessadas alternadamente, como mostra a Figura 12.

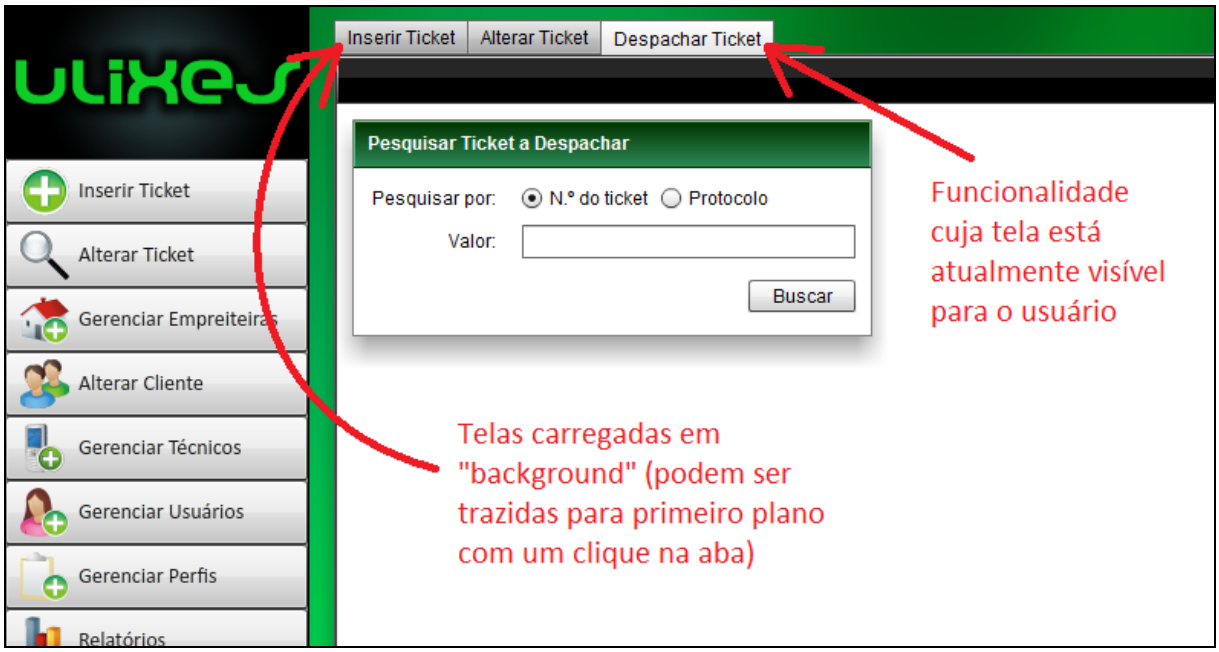


Figura 12 – Telas Abertas em Múltiplas Abas

Fonte: Autoria própria.

Caso se deseje fechar uma tela, basta trazê-la para o primeiro plano e utilizar o ícone em formato de "x" no canto superior direito (Figura 13).

Figura 13 – Ícone Para Fechamento de Tela

Fonte: Autoria própria.

Uma constante nas interfaces do sistema são os painéis tridimensionais com barra de título verde ao topo; este elemento gráfico foi amplamente empregado para agrupar dados relacionados, organizando melhor a grande quantidade de informação que era necessário fornecer em algumas ferramentas. A tela de consulta e alteração de tickets é um bom exemplo do uso criterioso de painéis, conforme ilustra a Figura 14.

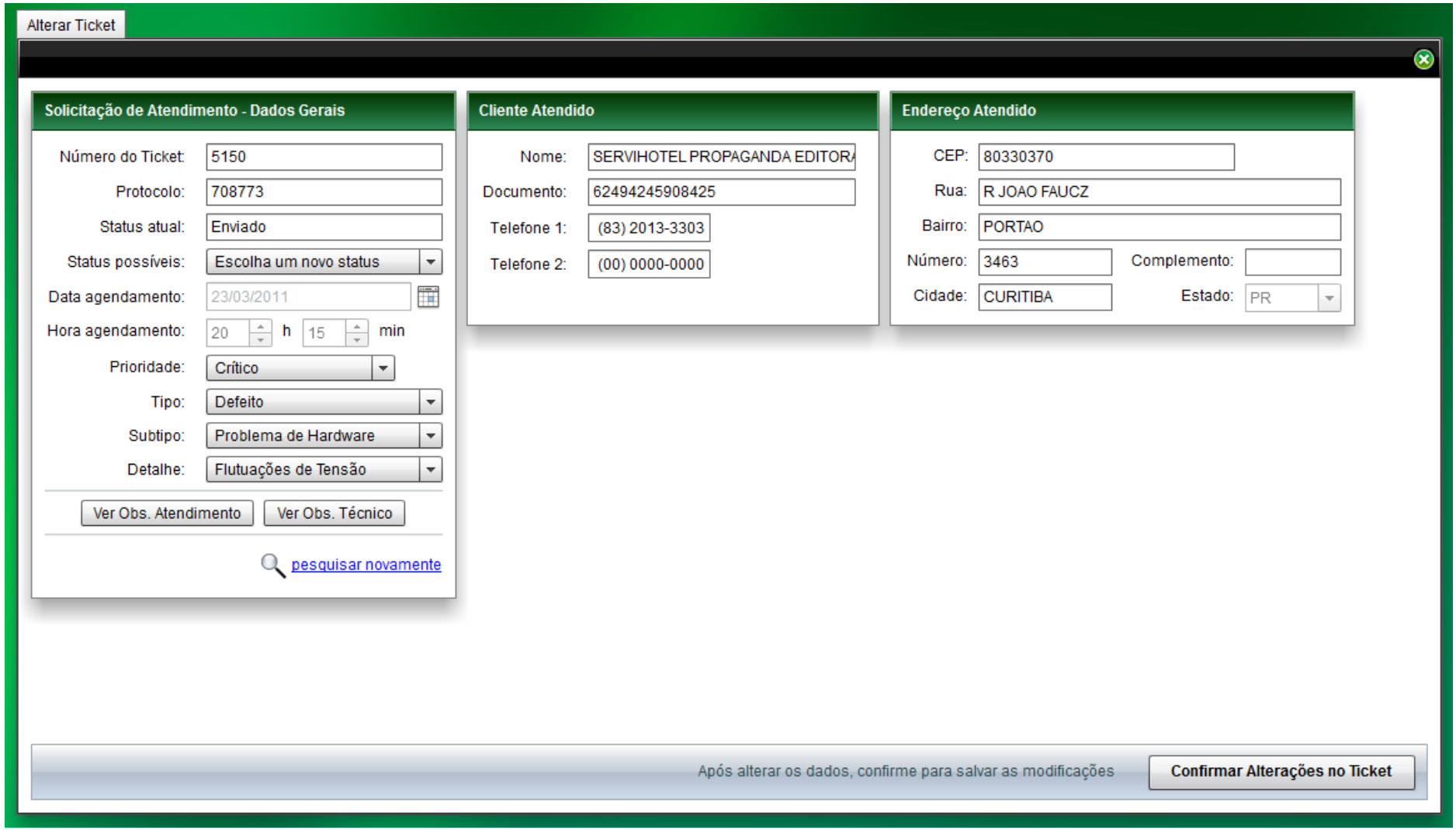


Figura 14 – Tela Alterar Ticket

Fonte: Autoria própria.

Os dados específicos da solicitação de atendimento estão reunidos no painel à extrema esquerda, tendo sido propositalmente colocados nesta posição – pois o sentido de leitura usado no Ocidente é da esquerda para a direita. Representando a essência do ticket, devem ser os primeiros a serem visualizados pelo usuário.

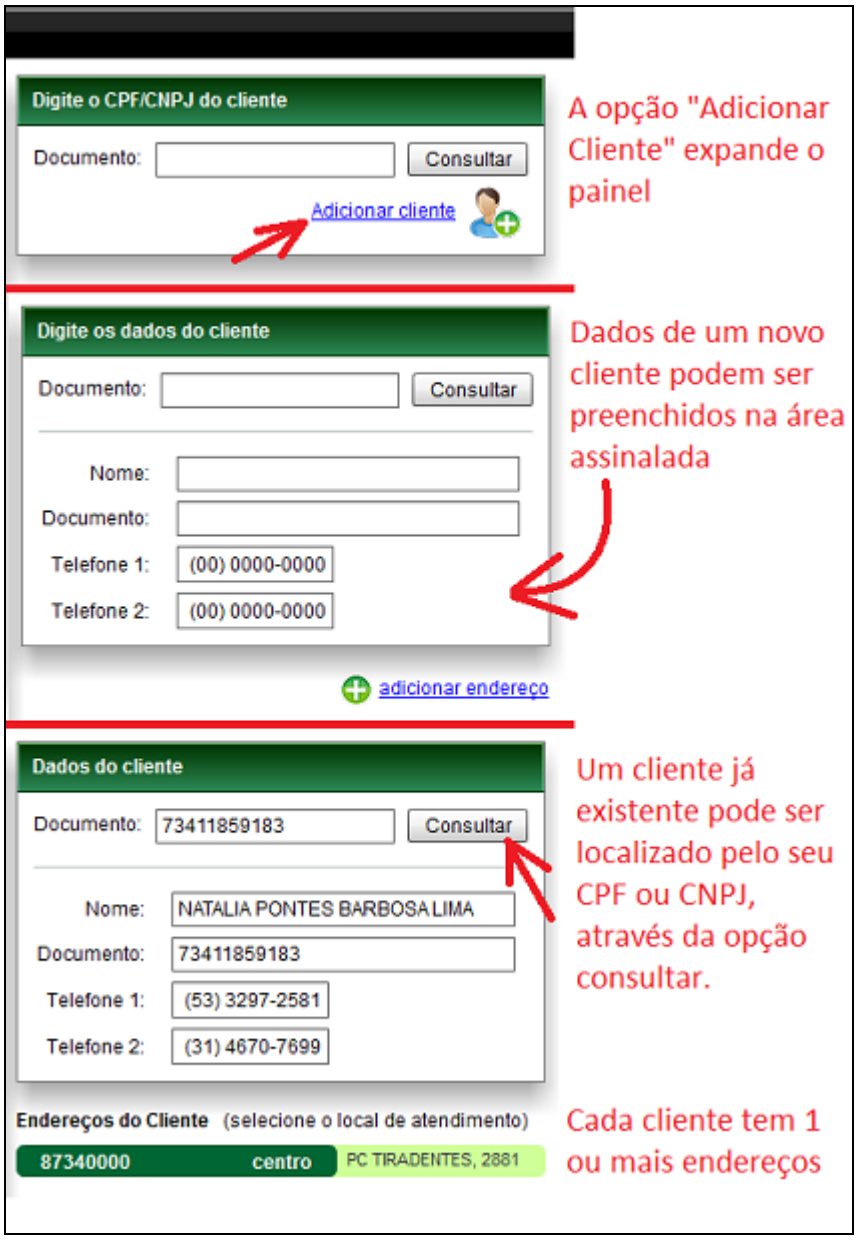


Figura 15 – Estados do Componente TelaCliente

Fonte: Autoria própria.

As informações do cliente e o endereço a ser atendido foram distribuídos em dois outros painéis separados. Estes módulos em especial (TelaCliente e TelaEndereco) servem não apenas para exibir dados, mas para permitir a realização de consultas no banco. Extremamente

utilitários, eles foram reaproveitados em diversas telas. O valor de um atributo booleano interno define se estarão ou não disponíveis para edição quando são instanciados; em telas como a de inserção de tickets, os painéis são usados em formato editável, oferecendo ao usuário certas facilidades (como busca de endereço por CEP e busca de cliente por documento). A cada etapa de um caso de uso, os painéis se transformam para apresentar apenas a quantidade de informação necessária à tarefa em andamento, de maneira a não confundir o usuário ou saturá-lo. A Figura 15 e a Figura 16 mostram a versatilidade destes módulos.

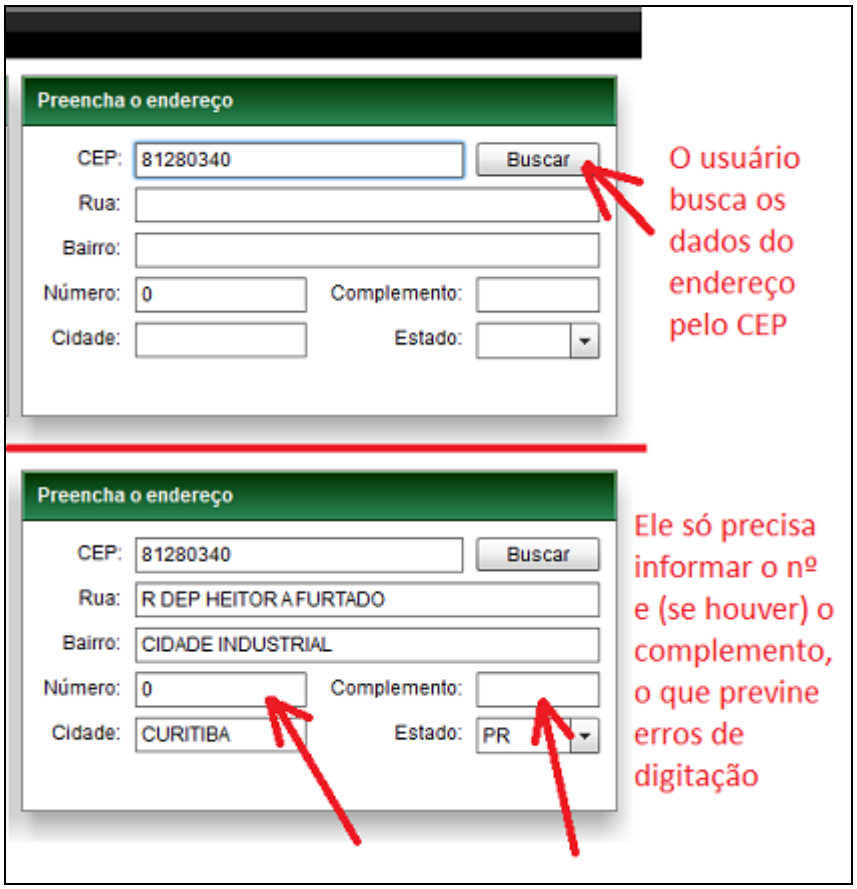


Figura 16 – Estados do Componente TelaEndereco
Fonte: Autoria própria.

Uma tela que merece especial destaque é a tela de extração de relatórios, pois ela utiliza um dos recursos mais bem-elaborados do Adobe Flex: seus componentes customizáveis para a geração de gráficos. O *framework* não apenas produz uma imagem com o gráfico desejado, mas também possibilita o tratamento de eventos relacionados à mesma – como, por exemplo, o ato de clicar em uma fatia específica de um gráfico de setores.

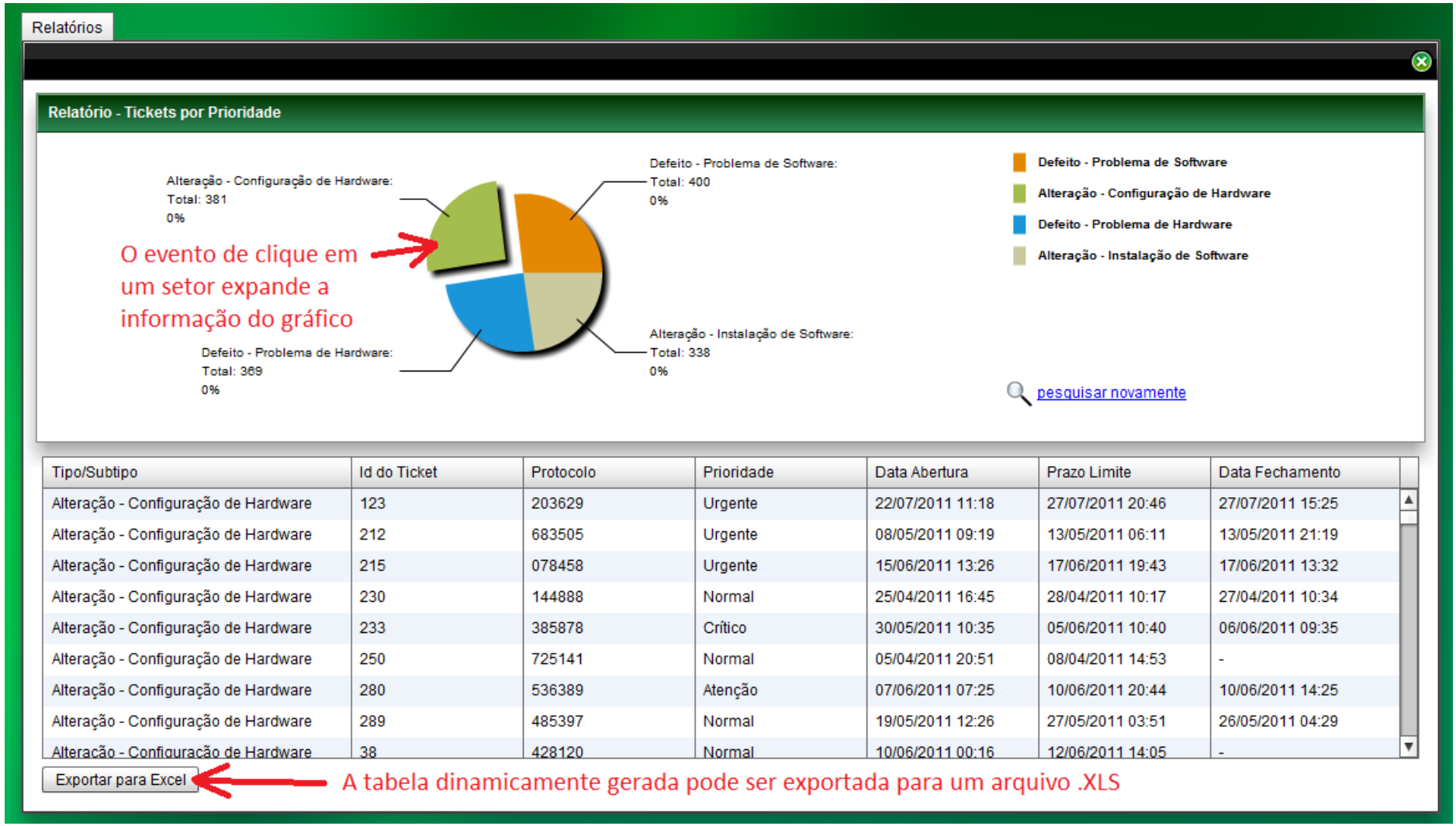


Figura 17 – Tela de Relatórios

Fonte: Autoria própria.

Na tela de “Relatórios” (vide Figura 17), o evento mencionado desencadeia uma nova busca ao banco, para detalhar os totais referenciados pelo gráfico em uma tabela de registros, sendo esta exportável para o Microsoft Excel, em formato de arquivo XLS.

Por fim, será feita uma apresentação breve das demais telas da aplicação nas páginas a seguir. Os detalhes do algoritmo de cada funcionalidade podem ser encontrados nos diagramas de seqüência correspondentes, caso o leitor deseje aprofundar-se em detalhes técnicos.

Na Figura 18, está representada a tela principal da funcionalidade “Inserir Ticket”, que em muito se assemelha à tela de alteração de tickets vista na Figura 14. As similaridades são propositais, e se destinam a criar uma identidade visual para o Ulixes, melhorando a usabilidade do *software* e tornando a navegação intuitiva e previsível.

Inicialmente, o usuário é solicitado a preencher as informações essenciais de ticket – classificação do problema, número de protocolo associado, data de agendamento e prioridade. O cliente vinculado ao ticket pode ser localizado na base de dados através do CPF/CNPJ, ou um novo cliente pode ser inserido. Qualquer uma das possibilidades habilita o painel de cliente, no centro da tela.

Uma vez definido o cliente, um endereço do mesmo pode ser selecionado (caso trate-se de uma pessoa já cadastrada no banco de dados) ou acrescentado (tratando-se de um novo cliente). Esse endereço é exibido no painel da extrema direita. A gravação das informações é disparada com um clique no botão “Confirmar e Gerar Ticket”, no canto inferior direito.

Se a adição de novos clientes é feita no ato de criação de um ticket, a edição de clientes já existentes na base do Ulixes ocorre em uma tela própria, mostrada na Figura 19. O cliente a ser alterado é resgatado em uma consulta cujo filtro é o CPF/CNPJ, e os dados pessoais – nome, número de documento e telefones – são apresentados no já familiar formato de painel.

Uma lista de endereços (um cliente pode ter de 1 a ‘n’ endereços distintos) é carregada logo abaixo, e ao clicar em qualquer dos itens da lista, detalhes são expandidos para um segundo painel no centro da tela. Endereços existentes não podem ser modificados, pois estão associados a tickets anteriores e é necessário preservar um histórico. Em situações de mudança de residência, porém, um novo endereço pode ser acrescentado. As alterações são salvas no banco após um clique no botão de confirmação, no canto inferior direito da janela.

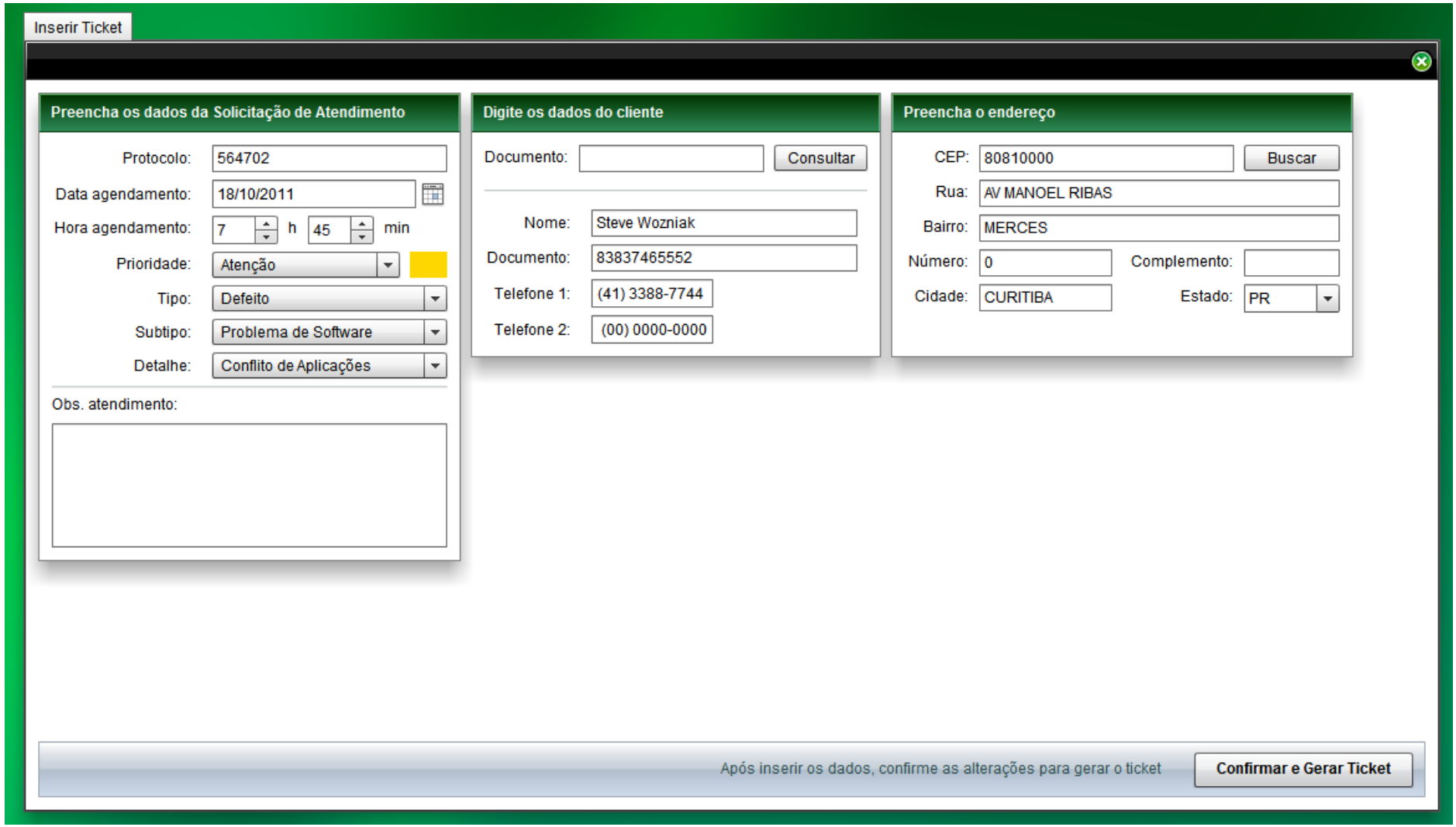


Figura 18 – Tela de Inserção de Tickets

Fonte: Autoria própria.

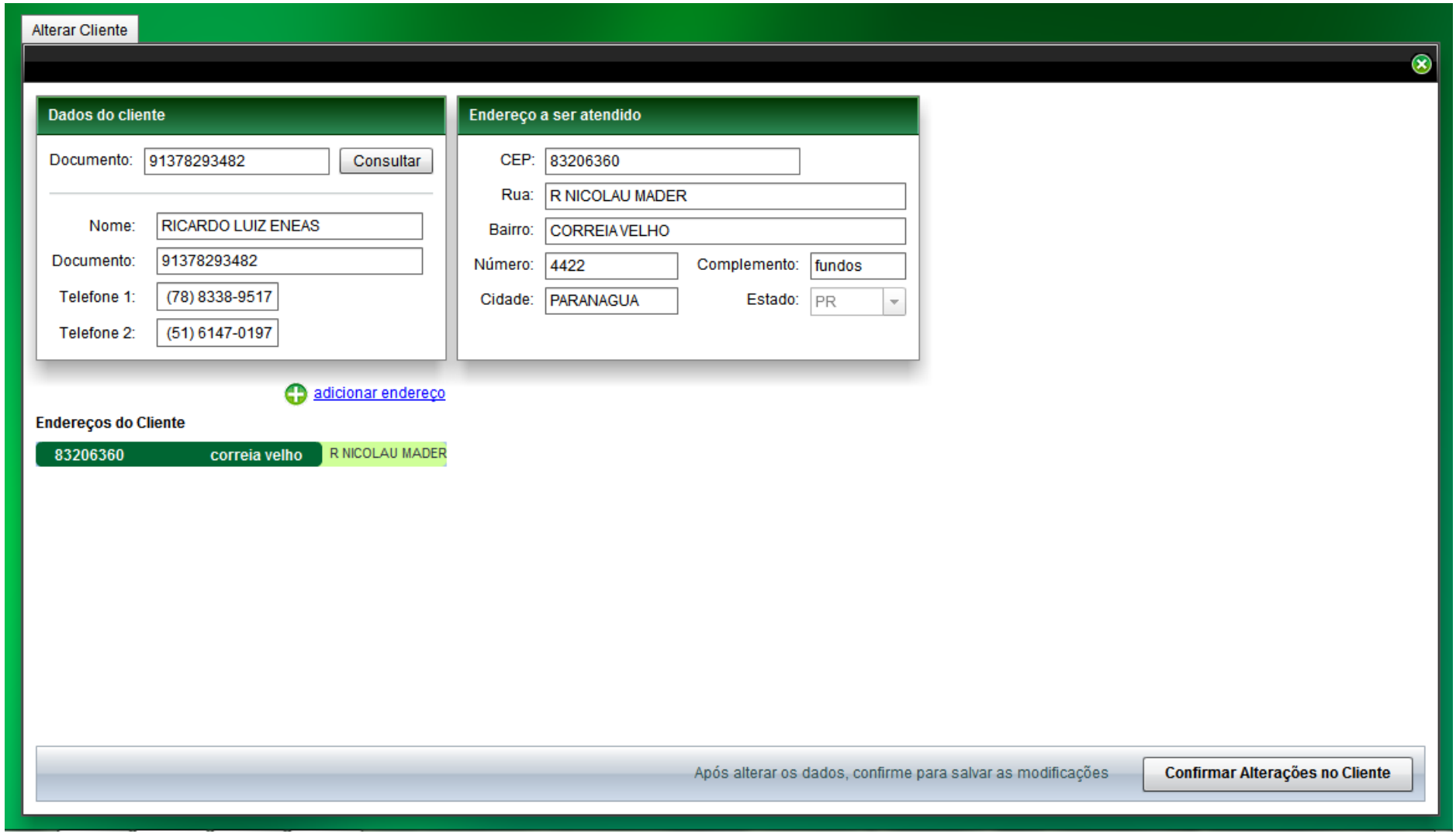


Figura 19 – Tela de Alteração do Cadastro de Clientes

Fonte: Autoria própria.

Por razões administrativas, foi decidido que além do processo automático de envio de tickets, seria útil – até mesmo necessário – que houvesse a possibilidade de despachar manualmente uma determinada solicitação para o celular de um técnico específico. Esta funcionalidade está retratada na Figura 20.

O ticket a ser disparado é localizado através de seu número, que deve ser inserido numa pequena janela de busca. Se o status do ticket não permitir sua emissão, uma mensagem de alerta será exibida para o usuário. Em caso contrário, todas as informações da solicitação de atendimento são trazidas na tela, em uma estrutura visual próxima à usada nas ferramentas de inserção e alteração de tickets.

O painel exclusivo desta funcionalidade, contudo, é o de seleção de empreiteira e técnico. Uma lista das empreiteiras é fornecida para que o usuário escolha a que deseja. Assim que é definida esta opção, os técnicos vinculados à empreiteira são apresentados em um segundo menu *drop-down*. Ao clicar sobre um técnico disponível, informações detalhadas sobre o mesmo preenchem o restante da janela. A solicitação é remetida apenas após um clique no botão "Despachar Ticket".

A Figura 21 apresenta a tela de gerenciamento de empreiteiras, que tem por objetivo agrupar os casos de uso 'cadastrar empreiteira' e 'alterar empreiteira'. Para que tal módulo esteja habilitado, é necessário que o usuário tenha perfil de administrador.

O processo de inserção de uma nova empreiteira e iniciado ao se clicar sobre o botão 'Inserir Nova Empreiteira', o qual abre um *popup* que requisita o preencimento dos seguintes dados pelo usuário: nome, CNPJ, um ou dois telefones comerciais desta empresa, CEP, número e complemento ao endereço desta empreiteira. Rua, bairro, estado e cidade podem ser preenchidos automaticamente utilizando-se o botão 'buscar', que consulta tais informações na base de dados usando o CEP como chave para tal pesquisa.

Para se alterar os dados cadastrais de uma determinada empreiteira, é necessário que o usuário clique duas vezes sobre uma das companhias listadas na tela de gerência de empreiteiras. Ao se fazer isso, um *popup* é aberto sobre essa página mostrando todos os dados desta empreiteira. Neste *popup*, todos os campos são editáveis.

Outro ponto importante a se comentar é que uma empreiteira pode ser desativada utilizando essa funcionalidade. Para isto, basta ao usuário clicar no checkbox 'empreiteira ativa', tirando a seleção deste campo, e em seguida salvar tal empresa, desta forma, esta empreiteira será ignorada nas rotinas de processamento de tickets.

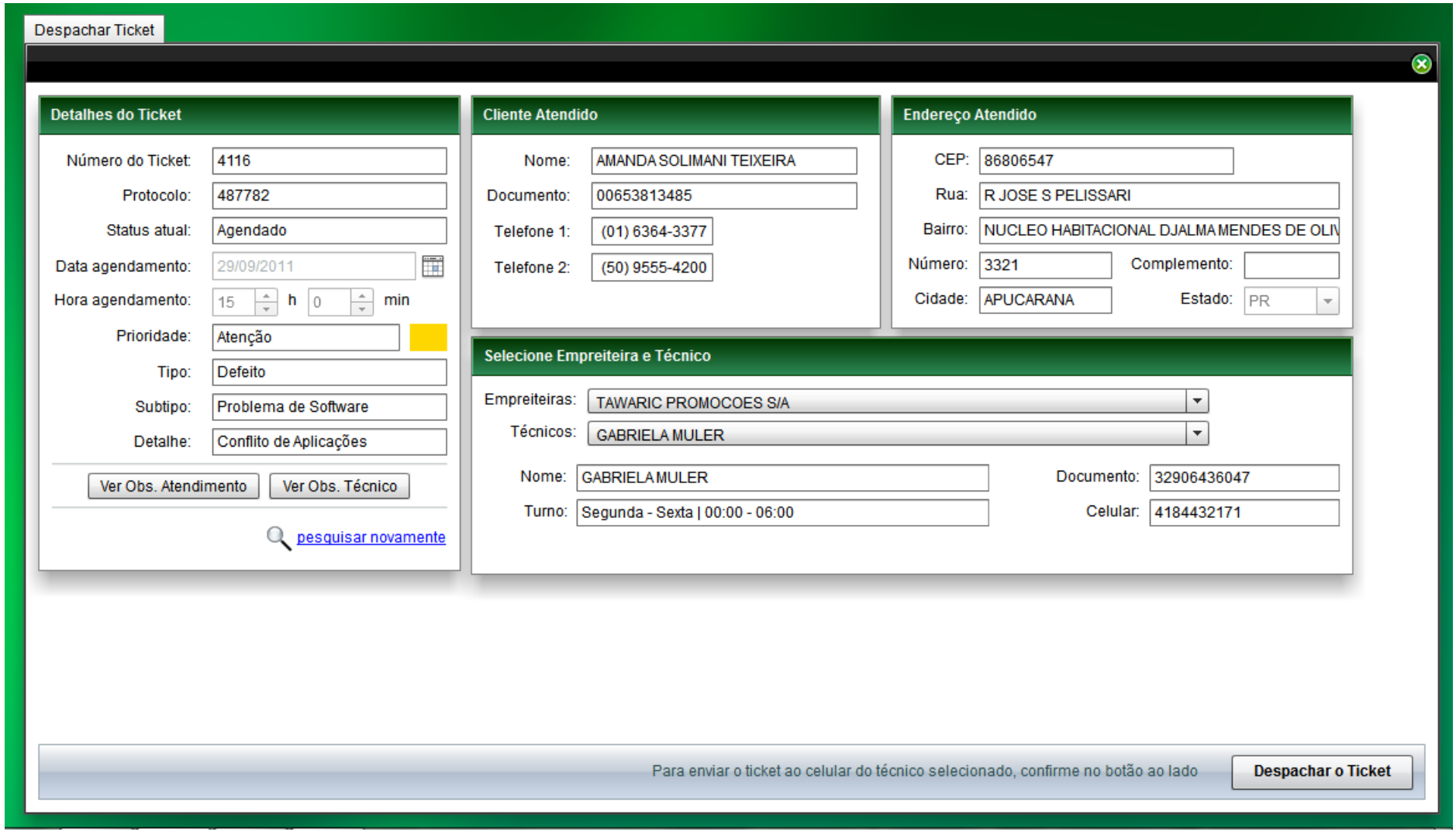
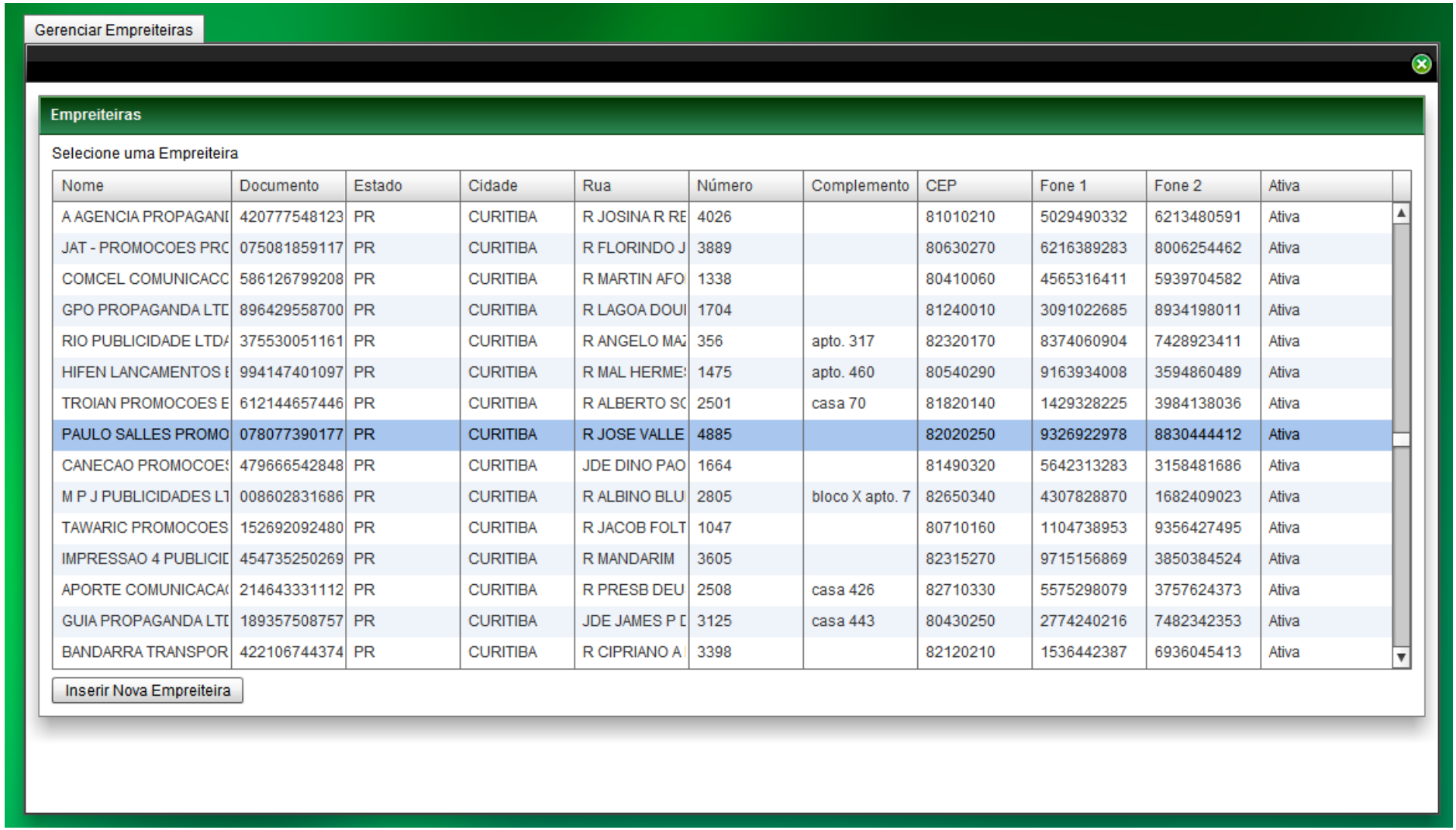


Figura 20 – Tela de Alteração do Cadastro de Clientes

Fonte: Autoria própria.



Nome	Documento	Estado	Cidade	Rua	Número	Complemento	CEP	Fone 1	Fone 2	Ativa
A AGENCIA PROPAGANI	420777548123	PR	CURITIBA	R JOSINA R RE	4026		81010210	5029490332	6213480591	Ativa
JAT - PROMOCOES PRC	075081859117	PR	CURITIBA	R FLORINDO J	3889		80630270	6216389283	8006254462	Ativa
COMCEL COMUNICACC	586126799208	PR	CURITIBA	R MARTIN AFO	1338		80410060	4565316411	5939704582	Ativa
GPO PROPAGANDA LTD	896429558700	PR	CURITIBA	R LAGOA DOUI	1704		81240010	3091022685	8934198011	Ativa
RIO PUBLICIDADE LTDA	375530051161	PR	CURITIBA	R ANGELO MAZ	356	apto. 317	82320170	8374060904	7428923411	Ativa
HIFEN LANCAMENTOS E	994147401097	PR	CURITIBA	R MAL HERME	1475	apto. 460	80540290	9163934008	3594860489	Ativa
TROIAN PROMOCOES E	612144657446	PR	CURITIBA	R ALBERTO SC	2501	casa 70	81820140	1429328225	3984138036	Ativa
PAULO SALLES PROMO	078077390177	PR	CURITIBA	R JOSE VALLE	4885		82020250	9326922978	8830444412	Ativa
CANECAO PROMOCOES	479666542848	PR	CURITIBA	JDE DINO PAO	1664		81490320	5642313283	3158481686	Ativa
M P J PUBLICIDADES LT	008602831686	PR	CURITIBA	R ALBINO BLU	2805	bloco X apto. 7	82650340	4307828870	1682409023	Ativa
TAWARIC PROMOCOES	152692092480	PR	CURITIBA	R JACOB FOLT	1047		80710160	1104738953	9356427495	Ativa
IMPRESSAO 4 PUBLICID	454735250269	PR	CURITIBA	R MANDARIM	3605		82315270	9715156869	3850384524	Ativa
APORTE COMUNICACAO	214643331112	PR	CURITIBA	R PRESB DEU	2508	casa 426	82710330	5575298079	3757624373	Ativa
GUIA PROPAGANDA LTD	189357508757	PR	CURITIBA	JDE JAMES P D	3125	casa 443	80430250	2774240216	7482342353	Ativa
BANDARRA TRANSPOR	422106744374	PR	CURITIBA	R CIPRIANO A	3398		82120210	1536442387	6936045413	Ativa

Figura 21 – Tela Gerenciar Empreiteira

Fonte: Autoria própria.

Conforme a tela da Figura 22, podemos verificar que a tela de gerenciamento de técnicos é dependente da funcionalidade de cadastro de empreiterias, pois para que um técnico seja atualizado ou inserido, e necessário primeiramente se escolher a empresa a qual este técnico está ou estará vinculado. Desta forma, um requisito necessário para a utilização desta funcionalidade e a inserção prévia das companhias que serão utilizadas para a inserção dos técnicos. Esta tela só está disponível para usuários com perfil de administrador.

Ao se clicar no botão ‘Inserir Técnico’ um novo painel é mostrado na parte inferior da tela, requisitando o nome, o telefone, o documento e o turno do novo técnico. Após inserir todas essas informações, o usuário deve clicar no botão ‘Inserir’. Se não ocorrer nenhum problema de validação, o novo técnico é inserido na base de dados.

Para se atualizar os dados cadastrais de um técnico previamente inserido, é necessário se clicar duas vezes sobre este técnico na tabela ‘Técnicos’ (vide Figura 22). Fazendo-se isso, um novo painel será mostrado ao usuário na parte inferior da tela, com todas as informações do usuário acessíveis à edição.

Após efetuar as atualizações necessárias, o usuário deve clicar no botão ‘Atualizar’ para persistir tais informações no banco de dados. Um aspecto importante a se ressaltar é que as mesmas validações aplicadas ao processo de inserção de técnicos são utilizadas para o processo de atualização, impedindo a inserção de dados incorretos. Um técnico pode ser desativado com a utilziação do *checkbox* ativo, tirando-se a seleção deste, e persistindo as alterações no banco de dados, o técnico é removido do processo de envio de tickets.

A funcionalidade representada pela Figura 23 permite o gerenciamento dos perfis que serão vinculados aos usuários. Com esta tela, é possível se definir os níveis de acesso que cada perfil de usuário terá no sistema.

O primeiro passo para a inserção de um novo perfil é o clique sobre o botão inserir na parte inferior da página de gerência de perfis. Após tal procedimento, um novo painel é apresentado na parte inferior da página, com o título “Adicionar Perfil”. Este painel requisita apenas um parâmetro, o nome do novo perfil, sendo que é necessário após o preenchimento deste que o usuário clique no botão “Inserir Funcionalidades” para que sejam selecionadas as telas as quais este perfil terá acesso.

No novo painel mostrado ao usuário após o último procedimento, existem campos de checagem vinculados a descrição da funcionalidade, que podem ser selecionados com um simples clique sobre o checkbox. Ao se clicar no ‘Inserir’, o processo de inserção de perfil é finalizado.

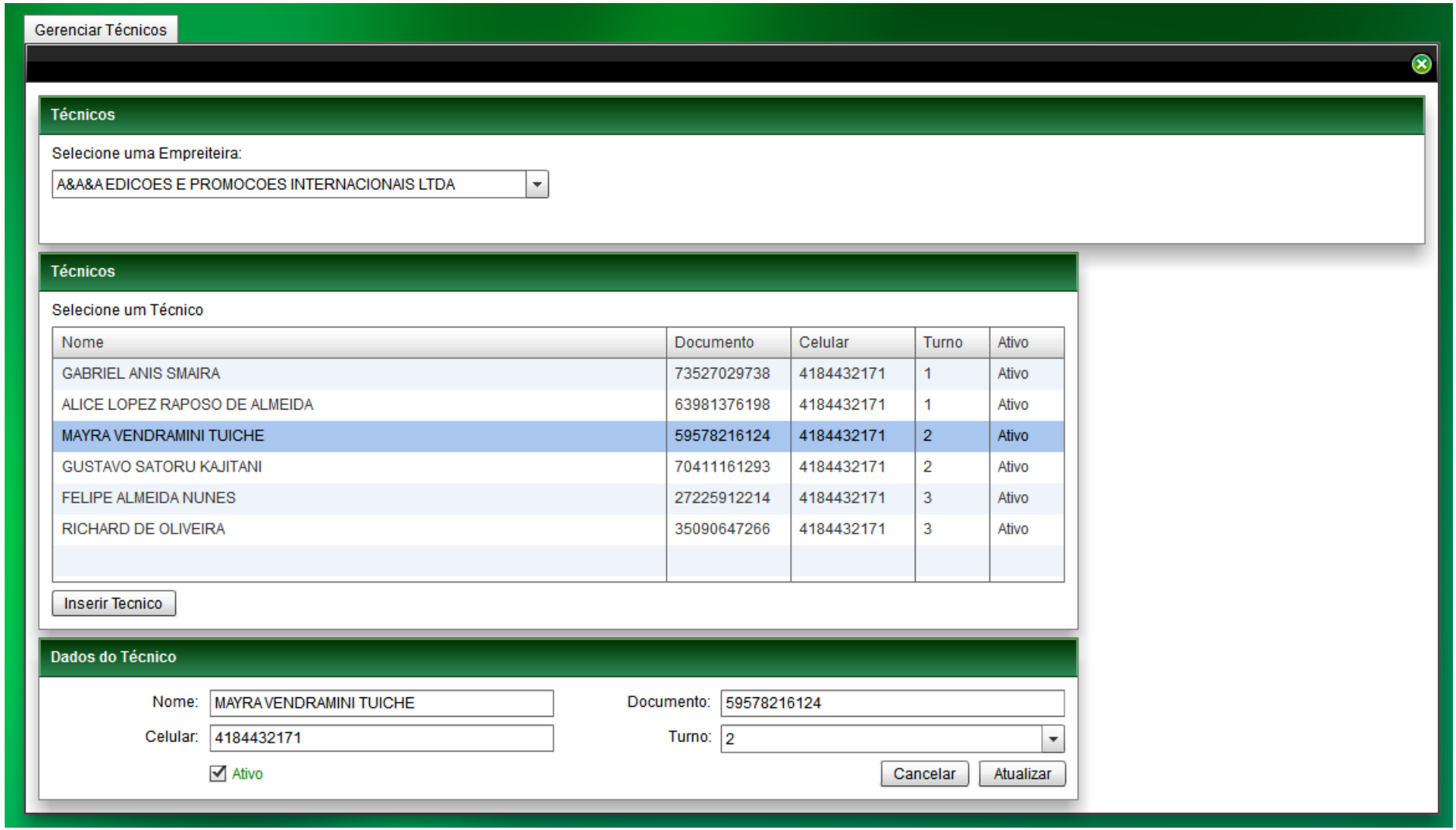


Figura 22 – Tela Gerenciar Técnicos

Fonte: Autoria própria.

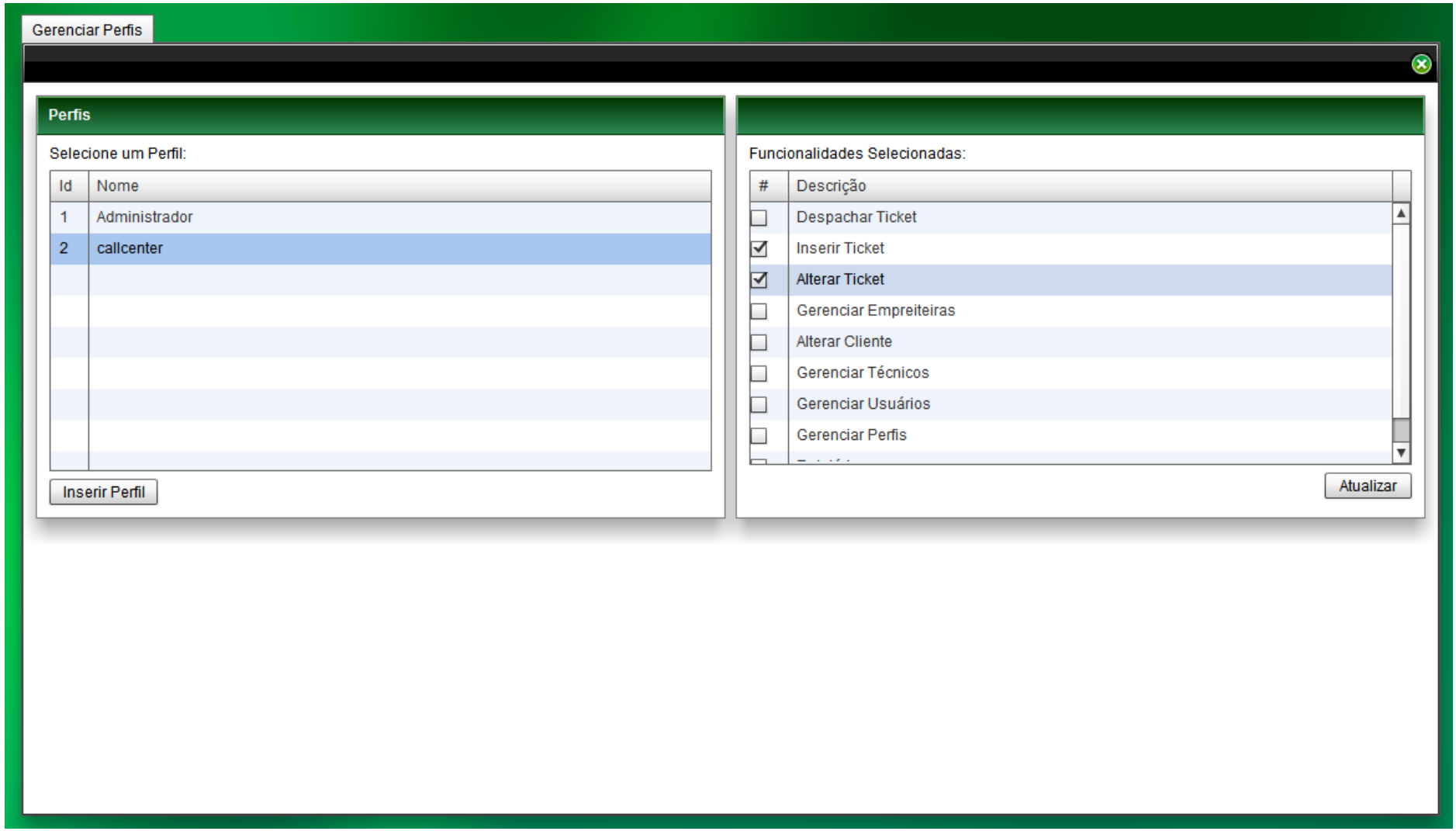


Figura 23 – Tela Gerenciar Perfis

Fonte: Autoria própria.

A atualização de perfis é um processo muito simples no sistema Ulixes. Para que o painel de edição de perfis seja habilitado basta ao usuário um clique simples sobre o nome do perfil que se deseja modificar.

Após esse primeiro passo, um painel é mostrado ao usuário, com todas as funcionalidades disponíveis no sistema e também apresentando as funcionalidades que foram selecionadas previamente para este perfil. Nesta tela, pode-se tanto adicionar novas funcionalidades quanto remover as que previamente foram atreladas a tal perfil. Na Figura 23 temos um exemplo desta funcionalidade sendo utilizada.

O último passo necessário ao processo de atualização de perfis é um clique sobre o botão “Atualizar”, que persistirá as novas funcionalidades selecionadas para esse perfil no banco de dados.

Para finalizar, na Figura 24 temos a tela de gerenciamento de usuários, na qual é possível se efetuar as operações de atualização e inserção de usuários.

O primeiro passo para a inserção de um novo usuário é abrir o *popup* responsável por esse processo. Isso pode ser feito clicando-se no botão “Inserir Usuário”.

No *popup* intitulado “Insira os Dados do Novo Usuário” é requisitado ao usuário o preenchimento da matrícula, da senha e do perfil do novo usuário. Esses dados são validados ao se clicar no botão Inserir e se nada de anormal for encontrado os dados são persistidos na base de dados.

O processo de atualização de usuários é bastante similar ao processo de atualização de empreiteiras. Para ter acesso a funcionalidade “Alterar Usuário”, basta um clique duplo sobre o nome do usuário que se deseja modificar na tela de gerência de usuários. Feito isso, um *popup* (vide figura 24) será disponibilizado para que os dados cadastrais do usuário do sistema possam ser alterados.

Estando nesse *popup*, todos os dados cadastrais do usuário são atualizáveis, sendo que as mesmas validações de dados aplicadas ao processo de criação de usuários são aplicadas ao processo de atualização destes. Outro ponto importante a se destacar é que um usuário pode ter seu acesso ao sistema removido por esse processo. Para isso, é necessário desmarcar a opção “Ativo” do formulário apresentado. Após os dados serem persistidos através de um clique sobre o botão “Atualizar”, a interface com o usuário é atualizada.

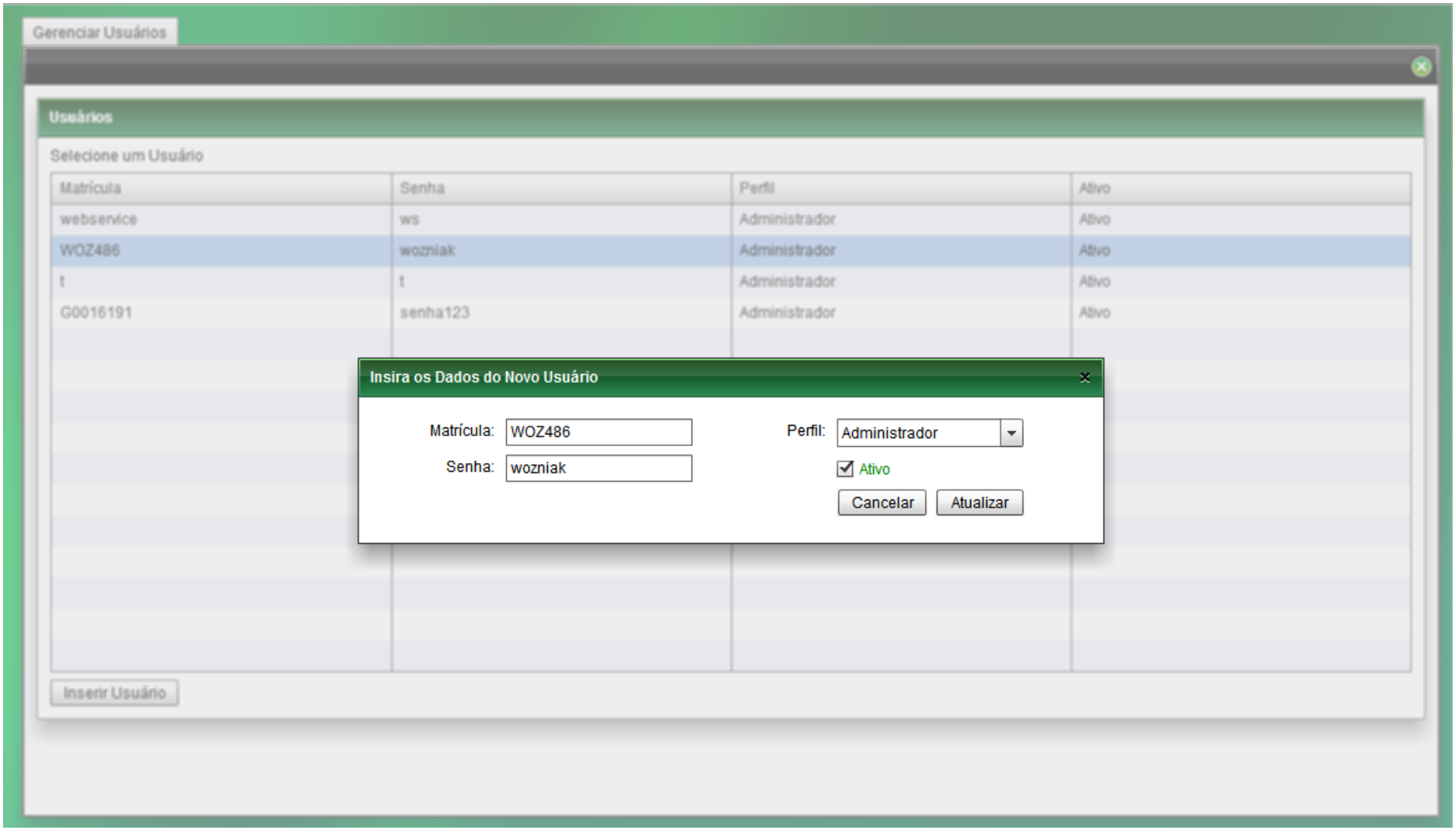


Figura 24 – Tela Gerenciar Usuários

Fonte: Autoria própria.

3.7.3 Envio de SMS

Sistemas corporativos que dependem do envio de mensagens SMS para dispositivos móveis costumam utilizar-se de serviços de *Gateway* SMS para a emissão de mensagens. Um serviço de *Gateway* SMS normalmente disponibiliza um número pré-definido de envios para o cliente, e também oferece uma interface para que a aplicação do contratante possa integrar-se ao seu próprio sistema e gerar as mensagens a serem mandadas. Essa interface em geral é um *web service*, que o cliente acessa através de uma rede externa, passando os parâmetros necessários à construção e à remessa do SMS. O emprego de um serviço de *Gateway* SMS é a solução mais usual para este tipo de cenário.

A contratação de um serviço dessa categoria, porém, implica em custos acima das possibilidades financeiras deste projeto. Se o Ulixes fosse comercializado, seria viável cogitar o uso de um *Gateway* SMS, mas em um primeiro momento foi preciso recorrer a uma solução mais econômica. Assim, decidiu-se implementar um mecanismo de envio de SMS dentro do próprio Ulixes.

No *Nokia Developer Wiki*, uma enciclopédia *online* para desenvolvedores interessados na plataforma Nokia, foram localizadas as informações necessárias para simular um equipamento profissional de envio de SMS. Através de um driver apropriado, um aparelho celular Nokia pode ser conectado a um computador através da porta USB, e comandos AT (*Attention Terminal*) de baixo nível podem ser enviados ao dispositivo, que então funcionará como um modem GSM.

O aparelho escolhido para a experiência foi um Nokia 5310 XpressMusic de propriedade de um dos integrantes da equipe. No próprio site brasileiro da Nokia está disponível para download o *driver* que permite ao sistema operacional reconhecer o dispositivo Nokia conectado à USB. Após a instalação do mesmo, o aparelho celular torna-se visível como um modem, conectado a uma porta serial virtual.

Os primeiros testes foram feitos com o auxílio do cliente Telnet/SSH conhecido como "Putty", que permite a abertura de conexões e o envio de *bytes* à porta serial de um computador. Comandos AT de discagem e também de envio de SMS foram mandados através do terminal do Putty, e o celular os interpretou corretamente.

O próximo passo foi encontrar uma solução que permitisse a uma aplicação Java contatar a portal serial. Como o *software* Java não tem acesso direto ao *hardware* - uma vez que a JVM interpõe-se entre a aplicação e o sistema operacional - uma biblioteca disponibilizada sob a licença GNU LGPL teve de ser empregada para realizar a tarefa. Essa

biblioteca é a RXTX, e utiliza JNI (*Java Native Interface*), um recurso que possibilita a comunicação do Java com bibliotecas externas, construídas com o código nativo do sistema. As classes e interfaces do RXTX são compatíveis com a CommAPI da Sun (atualmente descontinuada), de modo que um conhecimento básico da CommAPI é o bastante para que um desenvolvedor seja capaz de codificar usando RXTX.

Com o dispositivo móvel conectado ao computador, produziu-se um código para abrir a porta serial, converter comandos AT textuais em *bytes* e enviá-los ao aparelho. Esse processo funcionou a contento, apesar de alguns problemas eventuais de *timeout* de conexão. Mesmo sendo um artifício para resolver a impossibilidade de arcar com os custos de um serviço de *Gateway* SMS, a solução adotada permitiu que se testasse o fluxo de envio de tickets como um todo, desde a criação de uma solicitação de atendimento até a remessa da mesma para o celular de um suposto profissional técnico.

3.7.4 DaoFactory e Camada de Acesso ao Banco de Dados

Um dos requisitos do sistema Ulixes, desde o início do projeto, foi a portabilidade da aplicação. Tendo isso em mente, procurou-se um modo de estruturar o sistema que facilitasse ao máximo a futura adaptação a outros bancos de dados que não o Oracle, o SGBD escolhido para a implementação inicial.

Durante as pesquisas para o desenvolvimento do Ulixes, o padrão de projeto "*Factory Method*" (Gamma et al., 1994) pareceu uma resposta promissora para a questão. Esse padrão apresenta o conceito de uma fábrica de objetos (*Factory*), responsável por produzir instâncias sem expecificar a classe exata do objeto a ser instanciado. A criação de um objeto pode exigir processos complexos que não são da alçada do invocador do método, ou gerar quantidade significativa de código duplicado, ou necessitar ainda de informação inacessível a quem está solicitando o novo objeto, entre outros cenários. O padrão "*Factory Method*" se propõe a solucionar todas estas dificuldades, com a definição de um criador abstrato – uma classe que declara métodos para cada um dos tipos de objeto a serem gerados – e um criador concreto, que implementa estes métodos.

Para a necessidade do Ulixes, foi empregada uma simplificação do padrão. Elaborou-se a classe DaoFactory, que funciona como criador concreto, e a figura do criador abstrato foi dispensada. O aspecto mais importante do DaoFactory é a capacidade de acessar informação que não deveria estar ao alcance da camada de negócios, e a partir desta informação decidir

qual a classe do objeto a ser instanciado – uma das situações que o padrão de projeto "*Factory Method*" busca resolver.

Os métodos da classe DaoFactory retornam interfaces que representam *Data Access Objects*; antes de decidir qual objeto instanciar, a fábrica abre uma conexão com o banco de dados, e obtém um conjunto de metadados através dessa conexão. Uma informação disponível nestes metadados é o nome do produto/SGBD (Oracle, MySQL, PostgreSQL, etc). Com base nesse critério, e sem a necessidade de qualquer parâmetro adicional ou arquivo de configuraçào, o DaoFactory decide qual implementação de interface instanciar.

Por limitações de cronograma, foram criadas apenas as versões Oracle das classes correspondentes às interfaces fornecidas pelo DaoFactory. Assim, atualmente o sistema identifica o nome do SGBD, e se o mesmo contiver a palavra "Oracle", devolve um objeto adequado. Se em momento futuro fosse preciso migrar o sistema para uma base de dados diferente, porém, as alterações seriam mínimas. Bastaria adicionar o driver disponibilizado pelo proprietário, implementar as interfaces já existentes de maneira consistente com o banco escolhido, e modificar a DaoFactory para reconhecer este novo banco, criando os objetos adequados em resposta. As demais camadas da aplicação permaneceriam absolutamente intactas.

3.8 DISCUSSÃO

Neste capítulo, abordou-se a aplicação Ulixes em si, desde as etapas de projeto e o *design* da solução até o resultado final. Pontos de destaque do programa foram ressaltados, e algumas decisões de cunho técnico – e também o papel das tecnologias no desenvolvimento do Ulixes – foram examinados criticamente. Para manter a leitura dinâmica, incluiu-se no capítulo somente um dos diagramas de seqüência do *software*; caso haja interesse em conhecer os detalhes das outras funcionalidades, recomenda-se consultar o apêndice B, onde foi compilado e analisado o restante da documentação.

O próximo capítulo conclui este projeto, com uma avaliação do Ulixes do ponto de vista acadêmico. Possibilidades de trabalhos futuros também serão vistos no capítulo 4, a seguir.

4 CONCLUSÕES

Como uma aplicação pertencente à categoria dos sistemas de *Field Service Management* (FSM), o Ulixes foi concebido pensando-se na necessidade de uma empresa real.

O fluxo das atividades envolvidas, quando se gerencia um time de técnicos que presta atendimento em campo, é complexo o bastante para que o trabalho de automatizar esse fluxo seja válido, e o resultado, vantajoso. Entre as capacidades fundamentais de uma aplicação de FSM, devem estar o armazenamento dos dados relacionados aos clientes e às solicitações de atendimento, e a persistência segura destas informações.

O Ulixes adiciona a estes dois requisitos a possibilidade de remeter um *ticket* diretamente ao dispositivo móvel do técnico, tanto por um processo agendado quanto através da ação de um usuário. Favorecendo a versatilidade, o sistema oferece dois tipos de interface: um conjunto de telas através das quais um funcionário autorizado pode utilizar as diversas funcionalidades que a aplicação contém, e alguns *web services* que permitem a programas externos se comunicarem e realizarem operações dentro do sistema.

A implementação de um *software* com as características do Ulixes não teria sido possível, porém, sem a escolha criteriosa de algumas tecnologias, como a plataforma Java EE – empregada no *server-side* – e a plataforma Adobe Flex – amplamente utilizada na parte gráfica da aplicação. Como ambas funcionam sobre máquinas virtuais, a combinação delas conferiu ao Ulixes uma qualidade extra, altamente desejável: a portabilidade da aplicação.

Trabalhar com tecnologias tão consolidadas, que disponibilizam uma extensa gama de recursos ao desenvolvedor, foi uma experiência construtiva. O verdadeiro valor acadêmico do Ulixes, porém, não reside – pelo menos na opinião dos integrantes do projeto – em minúcias técnicas.

Antes de mais nada, o Ulixes é um sistema muito completo, pois combina em si mesmo vários conteúdos estudados durante o curso de Tecnologia em Sistemas para a Internet. Ele é uma aplicação *web*, e por isso demanda entendimento de servidores e protocolo http. É um *software* que se encaixa na categoria das RIA (*Rich Internet Aplications*), portanto exige interfaces com alto grau de interatividade e excelente usabilidade. O Ulixes gerencia grandes quantidades de informação, o que implica em um bom projeto de banco de dados, e em uma utilização inteligente do SGBD. O sistema faz o envio de mensagens SMS para telefones celulares, que devem estar preparados para receber esta informação – o que envolve

conhecimentos de codificação de baixo nível (para enviar os bytes ao aparelho) e de programação para dispositivos móveis. Finalmente, a aplicação dispõe de *web services*, servlets e de um EJB acionado através de um *Timer Service*, tecnologias consagradas de grande aceitação no mercado.

Por todos esses motivos, o Ulixes não só permitiu aos desenvolvedores exercitarem variados aspectos da informática em um único projeto, como também possibilitará que sejam avaliados em diversos quesitos. Mais do que um *software* de FSM, ele pode ser visto como um portfólio dos conhecimentos agregados durante o período de graduação, e do que é possível construir a partir deles. Este, sim, é o seu maior mérito.

4.1 TRABALHOS FUTUROS

Embora o Ulixes possa ser considerado um produto finalizado, ainda há espaço para o aperfeiçoamento da aplicação. Melhorias sugeridas são:

- Tela para cadastro de novas classificações de ticket (árvores de tipo, subtipo e detalhe de solicitação técnica);
- Tela para cadastro de novas prioridades de ticket;
- Implementações de classes de DAO para utilização de outros bancos de dados que não o Oracle (a estrutura do sistema foi previamente planejada para isso, e possibilita facilmente tal alteração);
- Novos formatos de relatório.

5 REFERÊNCIAS

5.1 BIBLIOGRAFIA

ARMSTRONG, Peter. **Hello Flex 4**. Greenwich: Manning Publications Co., 2010.

BUSCH, Marianne; KOCH, Nora. **Rich Internet Applications, State of the Art.** Munique: Universidade de Munique, 2009.

DEITEL, Harvey; DEITEL, Paul. **Java How to Program.** 6.ª ed. New Jersey: Prentice Hall, 2004.

FAIN, Yakov; RASPUTNIS, Victor; TARTAKOVSK, Anatole. **Rich Internet Applications With Adobe Flex and Java.** Woodcliff Lake: SYS-CON Media, 2007.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software.** Boston: Addison-Wesley Professional, 1994.

GASSNER, David. **Flash Builder 4 and Flex 4 Bible.** Indianapolis: Wiley Publishing, 2010.

KORE, Satish. **Flex 3 With Java.** Birmingham: Packt Publishing, 2009.

MCCUNE, Doug; SUBRAMANIAN, Deepa. **Adobe Flex 3.0 For Dummies.** Indianapolis: Wiley Publishing, 2008.

PRESSMAN, Roger S.. **Engenharia de Software.** 6.ª ed. Nova Iorque: McGraw-Hill, 2006.

TIWARI, Shashank. **Professional BlazeDS: Creating Rich Internet Applications.** Indianapolis: Wiley Publishing, 2009.

VIERIU, Valentin; TUICAN, Catalliin. **Adobe AIR, Bringing Rich Internet Applications to the Desktop.** Timişoara: Universidade de Timişoara, 2009.

5.2 MATERIAL *ONLINE*

Cloudhopper Opensource Lab RXTX Binary Distribution Website. Disponível em: http://www.cloudhopper.com/opensource/rxtx/. Acesso em: 9 Jun, 2011.

NOKIA**. Drivers para os cabos Nokia DKU-2, DKE-2, CA-42, CA-53, CA-70, CA-101 e CA-126.**
Disponível em: http://www.nokia.com.br/suporte-e-software/suporte-a-produtos/drivers-para-cabos/drivers-para-os-cabos-nokia-dku-2-dke-2-ca-42-ca-53-ca-70-ca-101-e-ca-126. Acesso em: 8 Jun, 2011.

NOKIA. **Nokia Developer Wiki - AT Commands.**
Disponível em: http://www.developer.nokia.com/Community/Wiki/AT_Commands. Acesso em: 8 Jun, 2011.

ORACLE. **The Java EE 6 Tutorial.**
Disponível em: http://download.oracle.com/javaee/6/tutorial/doc/. Acesso em: 21 Jul, 2011.

RXTX Wiki. Disponível em: http://rxtx.qbang.org/wiki/index.php/Main_Page. Acesso em: 9 Jun, 2011.

TATHAM, Simon; DUNN, Owen; HARRIS, Ben; NEVINS, Jacob. **PuTTY: A Free Telnet/SSH Client.**
Disponível em: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. Acesso em: 8 Jun, 2011.

APÊNDICE A – QUESTIONÁRIO DE LEVANTAMENTO DE REQUISITOS

QUESTÕES TÉCNICAS

1) Deve existir um banco de dados que armazene informações, ou será utilizado um arquivo?

Por motivos de segurança e também de facilidade de manutenção, será empregado um sistema gerenciador de banco de dados.

2) Utilizando-se um sistema gerenciador de banco de dados, qual é o sistema de preferência do cliente?

O sistema gerenciador de banco de dados utilizado inicialmente será o Oracle, contudo, a implementação deverá ser flexível para aceitar outros tipos de sistemas gerenciadores de banco de dados no futuro.

3) O sistema será desktop ou web?

O sistema deverá ser web, além de disponibilizar *web services* para facilitar a integração com outros sistemas.

4) Existe uma preferência por linguagem de implementação? Em caso positivo, qual?

Não existe preferência por uma linguagem específica, desde que a escolhida atenda às necessidades definidas para este projeto. Entretanto, um dos requisitos para o projeto é que este tenha uma interface amigável ao usuário, de preferência com a utilização de elementos de interface rica para internet (*RIA – Rich Internet Applications*).

5) O software será executado em um sistema operacional específico, ou a meta é algo versátil?

Um dos principais requerimentos do software é que ele seja versátil e possa ser utilizado o mais independentemente possível da plataforma, no caso de usuário (utilização como cliente), contudo podem-se considerar restrições do lado do servidor.

6) O sistema deverá levar em conta restrições de desempenho de *hardware*?

O sistema deverá funcionar de forma satisfatória em máquinas legadas.

7) Os dados a serem armazenados no Banco de Dados deverão ser criptografados?

a) Sim.

b) Não.

c) Não sei responder.

Resposta: B.

8) O sistema deverá ter um *log* de erros?

Sim, pois tal procedimento torna a correção de erros um processo muito mais barato e eficiente.

QUESTÕES OPERACIONAIS

1) Qual o esquema de cores a ser usado nas interfaces com o usuário?

Não existe um esquema de cores pré-definido; o importante é que a interface seja atrativa e não seja cansativa. Cores muito vivas, se utilizadas, deveriam ser dosadas com cautela.

2) Qual é o nível de experiência de usuário recomendada para utilização deste software?

a) Usuários iniciantes.

b) Usuários intermediários.

c) Usuários avançados.

d) Não sei responder.

Resposta: A.

3) Qual o tempo de vida útil de tal projeto?

Em torno de dez anos.

QUESTÕES ORGANIZACIONAIS

1) O sistema será empregado apenas em uma intranet, ou através da internet também?

O sistema deve estar disponível na intranet, visto que é um sistema de gerenciamento interno. Contudo, haverá certa interação dos técnicos de campo com o ambiente de forma remota (apenas atualizações de status).

2) Existiram privilégios para determinados usuários?

Sim, o acesso às funções administrativas será restrito a poucos usuários.

3) Que benefícios adicionais o sistema deve trazer em relação aos clientes?

O sistema deverá ser suficientemente ágil e estável para garantir atendimentos rápidos e eficientes aos clientes, de forma a cativá-los.

4) O sistema deverá gerar relatórios?

Sim, o sistema deverá gerar relatórios de atividades e de chamados.

5) O sistema deverá ter um *log* de operações?

Sim, pois isso tornará possível o rastreamento de eventuais problemas operacionais.

QUESTÕES ECONÔMICAS

1) O ambiente de desenvolvimento utilizado para o projeto será gratuito?

Sim, deverão ser utilizadas apenas ferramentas livres ou licenciadas para estudantes, ficando a cargo do comprador do produto a obtenção das licenças definitivas.

2) Qual a expectativa de entrega do sistema?

O sistema deverá ser entregue em cerca de oito meses (negociável).

3) Quem será responsável por efetuar manutenção no sistema, depois de entregue?

A equipe de desenvolvimento, por um tempo ainda a determinar.

4) Algum novo hardware será adquirido para este projeto?

As compras de novos equipamentos deverão ser evitadas, contudo, poderão ser efetuadas em casos extremos.

APÊNDICE B – DIAGRAMAS DE SEQÜÊNCIA

DIAGRAMA DE SEQÜÊNCIA BUSCAR DADOS TELA EMPREITEIRA

O diagrama a seguir mostra as interações entre as instâncias das classes responsáveis pela busca dos dados necessários à tela de gerência de empreiteiras. Tal diagrama está incluso tanto no diagrama de seqüência cadastrar empreiteira quanto no diagrama de seqüência alterar empreiteira.

Conforme mostra a Figura 25, este caso de uso representa basicamente a busca das informações relativas à empreiteiras no banco de dados e o seu transporte em objetos de transferência até a interface.

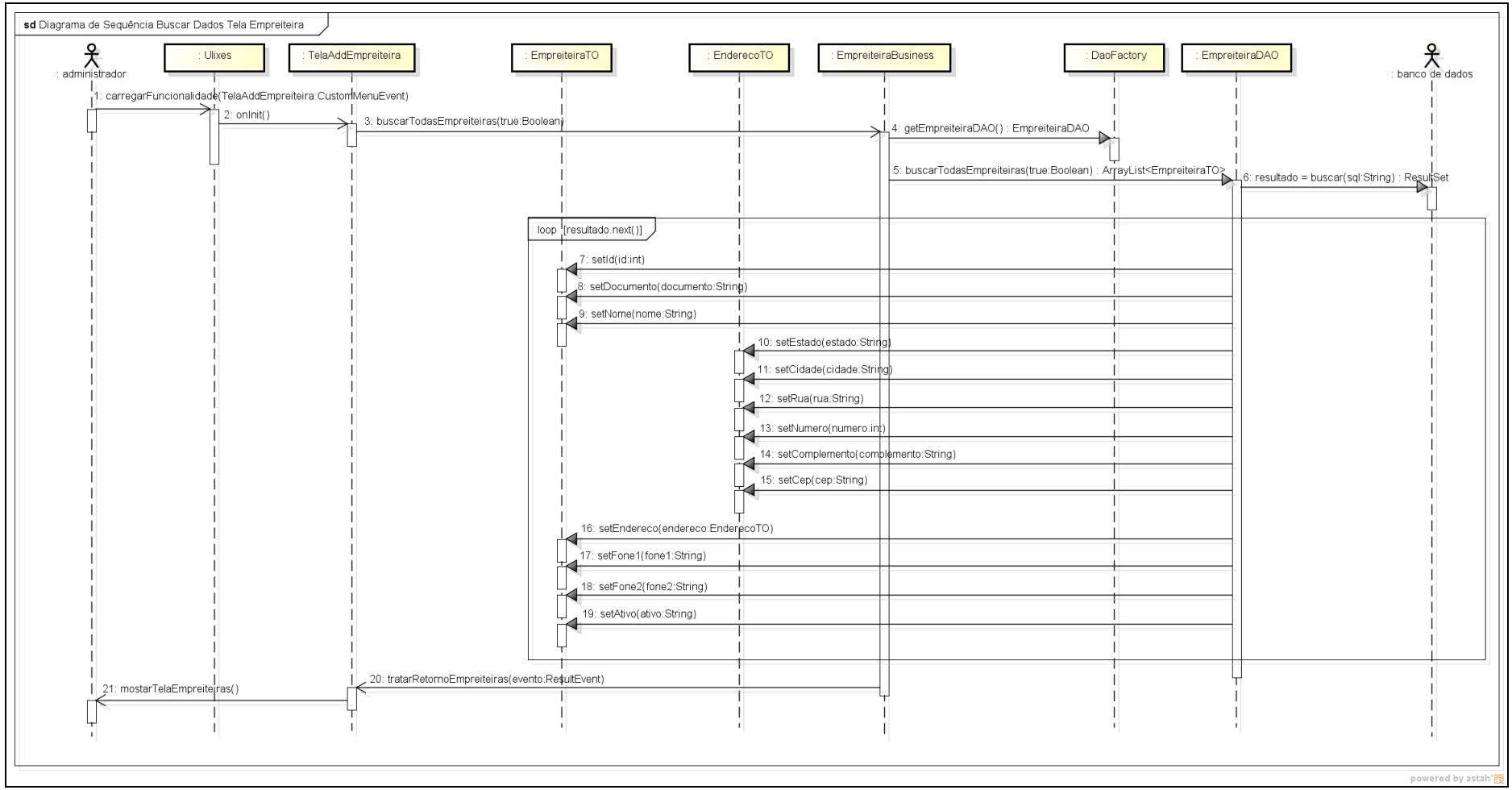


Figura 25 – Diagrama de Seqüência Buscar Dados Tela Empreiteira

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA CADASTRAR EMPREITEIRA

Neste diagrama são descritas todas as interações entre as instâncias de classes necessárias para o cadastro de empreiteiras no sistema. Para que tal módulo esteja habilitado, é necessário que o usuário tenha perfil de administrador.

A Figua 26 e a Figura 27 descrevem esse processo de inserção de empreiteiras na base de dados. Durante o processo de cadastro de uma nova empreiteira, é necessário que o usuário preencha o nome, o CNPJ, um ou dois telefones comerciais desta empresa, o CEP, o número e o complemento ao endereço desta empreiteira. Rua, bairro, estado e cidade podem ser preenchidos automaticamente utilizando-se o botão ‘buscar’, que consulta tais informações na base de dados usando o CEP como chave para tal pesquisa.

Por fim, também é necessário que o usuário indique uma área de cobertura para tal empreiteira antes de finalizar a inserção desta. Este é um passo muito importante, pois determinará quais tickets poderão ser encaminhados para técnicos desta empreiteira.

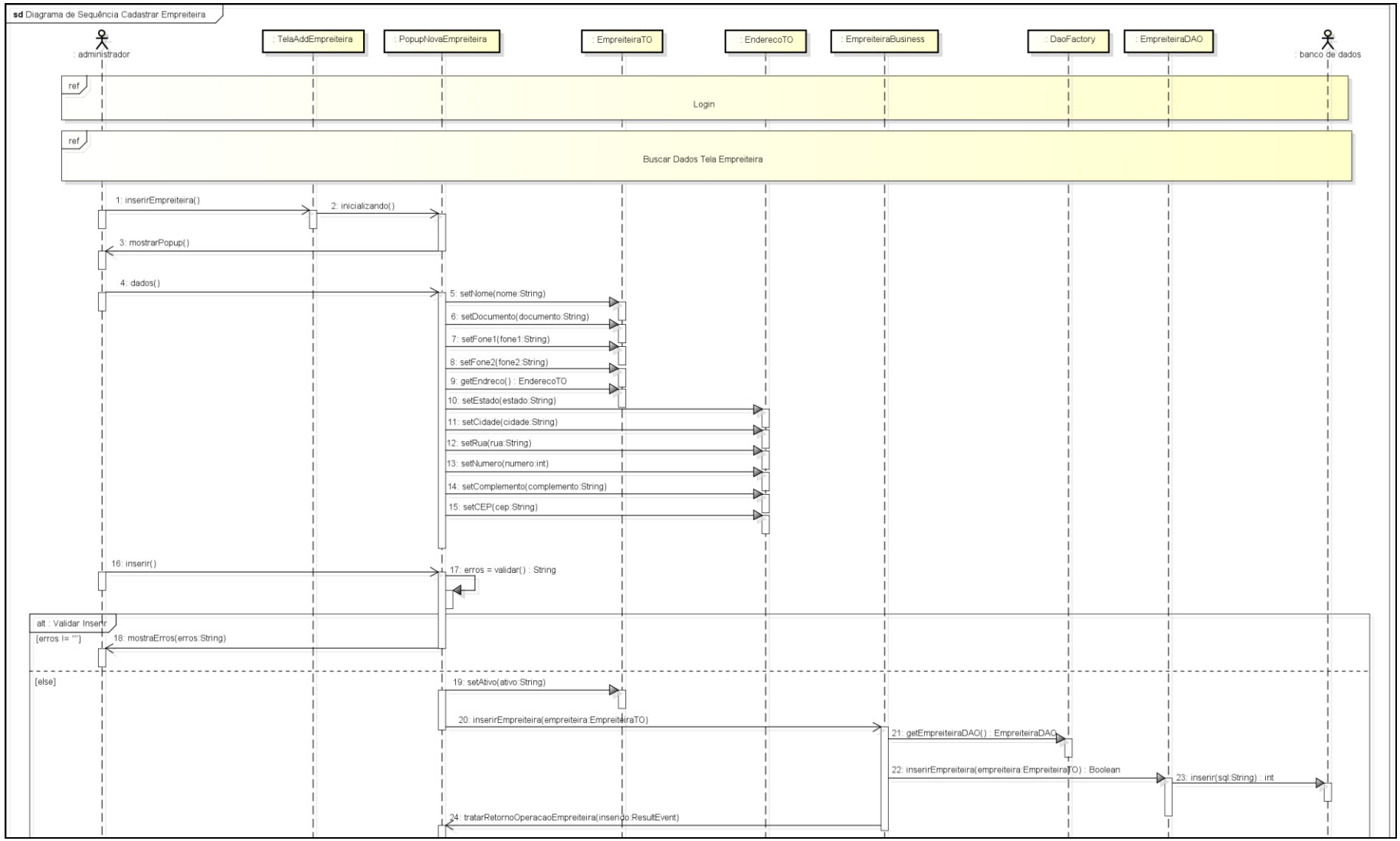


Figura 26 - Parte 1: Diagrama de Seqüência Cadastrar Empreiteira

Fonte: Autoria própria.

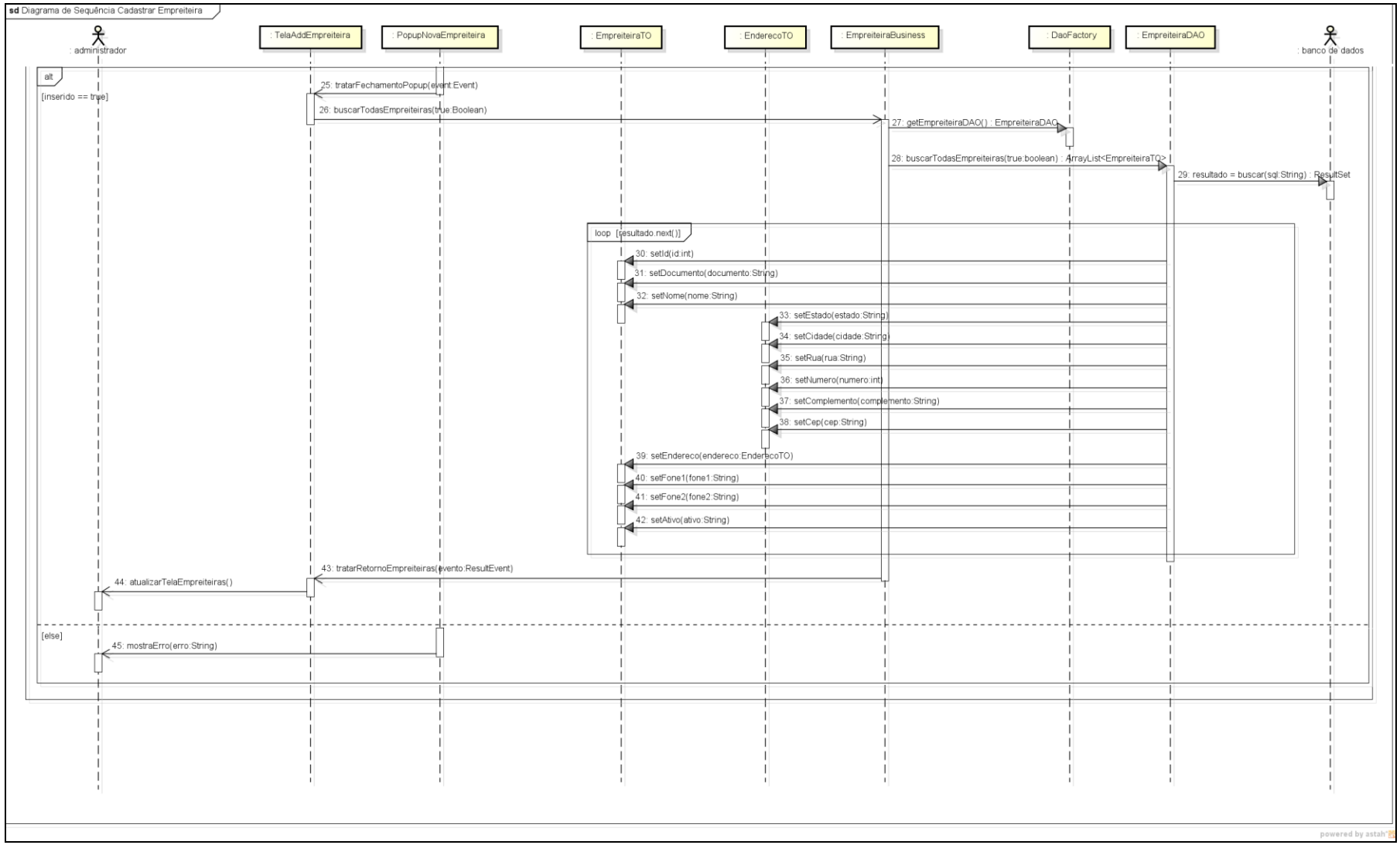


Figura 27 - Parte 2: Diagrama de Seqüência Cadastrar Empreiteira

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA ALTERAR EMPREITEIRA

O diagrama de seqüência alterar empreiteira descreve as interações entre classes necessárias para que um usuário com perfil de administrador do sistema possa alterar os dados cadastrais de uma empreiteira. Esse diagrama de seqüência utiliza as mesas classes que o diagrama de seqüência cadastrar empreiteira.

A Figura 28 e a Figura 29 ilustram o processo de modificação dos dados de uma empreiteira. Ao se clicar duas vezes sobre uma empreiteira na tela de gerência de empreiteiras, um *popup* é aberto sobre essa página mostrando todos os dados relativos a essa empreiteira. Nessa tela, todos os campos são editáveis.

Outro ponto importante a se comentar é que uma empreiteira pode ser desativada utilizando essa funcionalidade. Para isto, basta ao usuário clicar no checkbox ‘empreiteira ativa’, tirando a seleção deste campo, e em seguida salvar tal empreiteira, desta forma, esta empreiteira será ignorada nas rotinas de processamento de tickets.

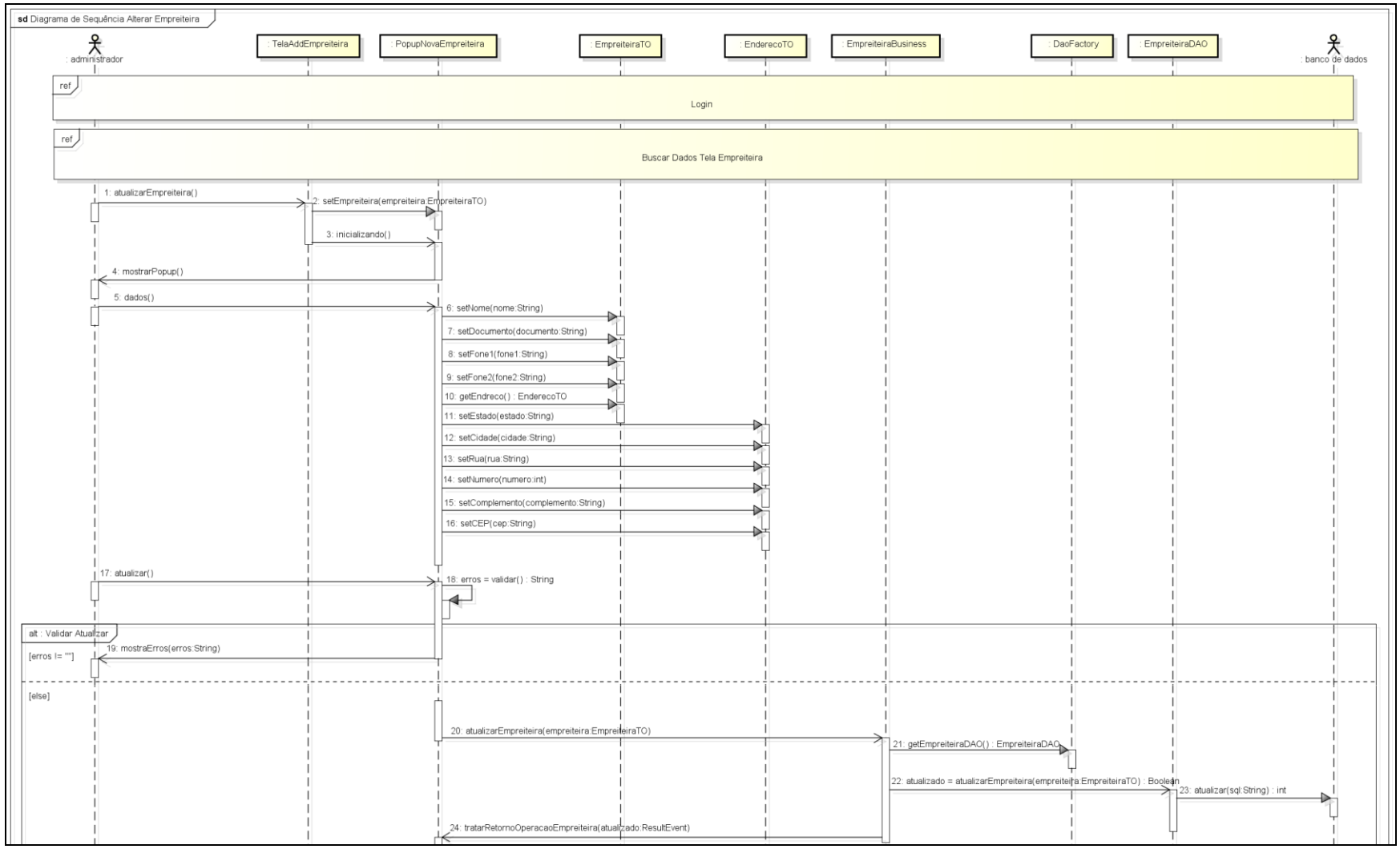


Figura 28 - Parte 1: Diagrama de Seqüência Alterar Empreiteira

Fonte: Autoria própria.

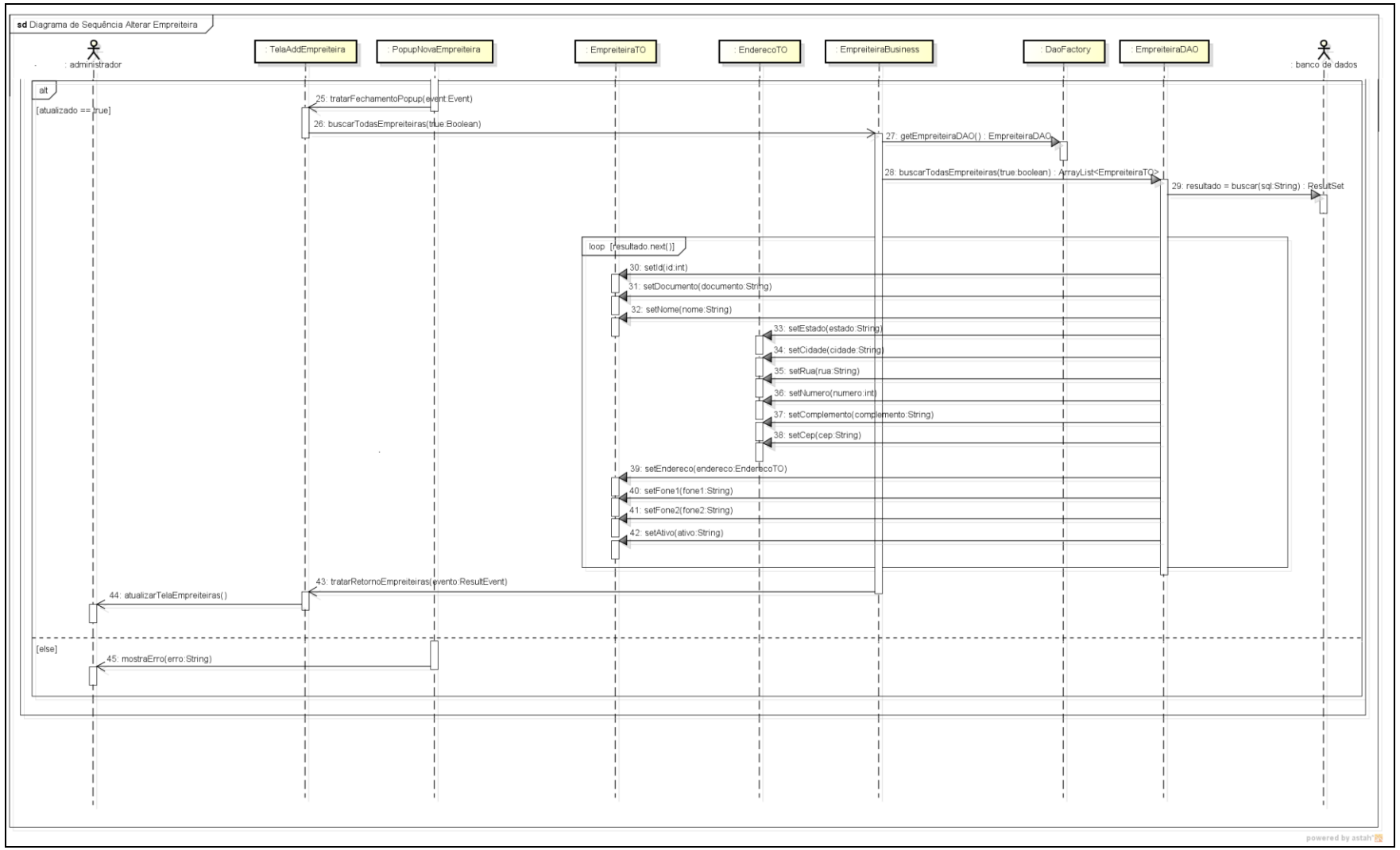


Figura 29 - Parte 2: Diagrama de Seqüência Alterar Empreiteira

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA BUSCAR DADOS TELA TÉCNICO

Assim como no caso do diagrama de seqüência “Buscar Dados Tela Empreiteira”, o diagrama de seqüência “Buscar Dados Tela Técnico” tem como função descrever as interações responsáveis pela busca dos dados necessários à operação de gerência de técnicos.

Este subcaso de uso é utilizado tanto no diagrama de seqüência “Cadastrar Técnico” quanto no diagrama de seqüência “Alterar Técnico”, visto que ambos requerem as mesmas informações.

Ao se observar a Figura 30 e a Figura 31, nota-se que este caso de uso basicamente descreve o processo de consulta ao banco de dados através de uma instância da classe abstrata “EmpreiteiraDAO”, a transferência destes dados até o objeto da classe de interface com o usuário “TelaAddTécnico” e o processo de apresentação de tais dados. Para a transferência de dados são utilizadas instâncias das classes “TurnoTO”, “TecnicoTO” , “EmpreiteiraTO” e “EnderecoTO”.

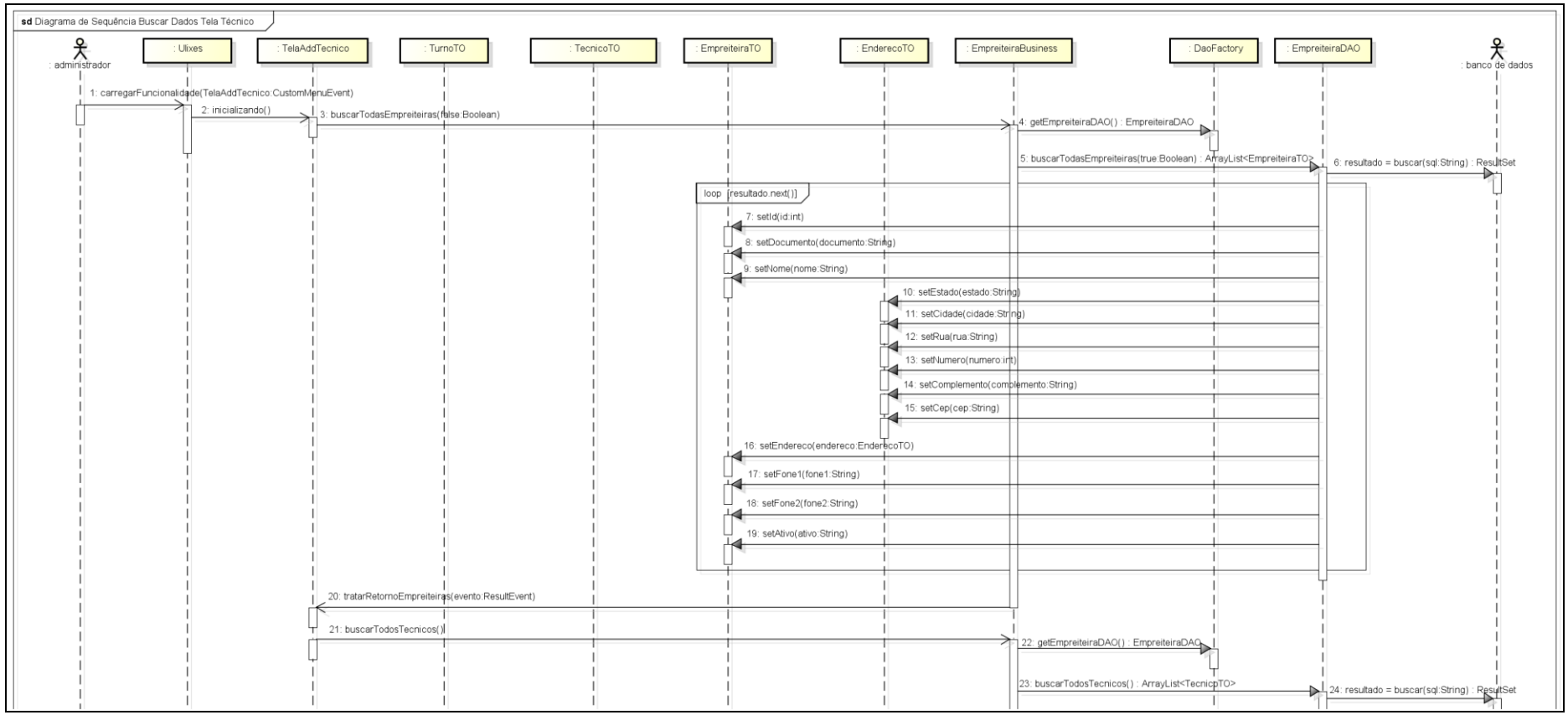


Figura 30 - Parte 1: Diagrama de Seqüência Buscar Dados Tela Técnico

Fonte: Autoria própria.

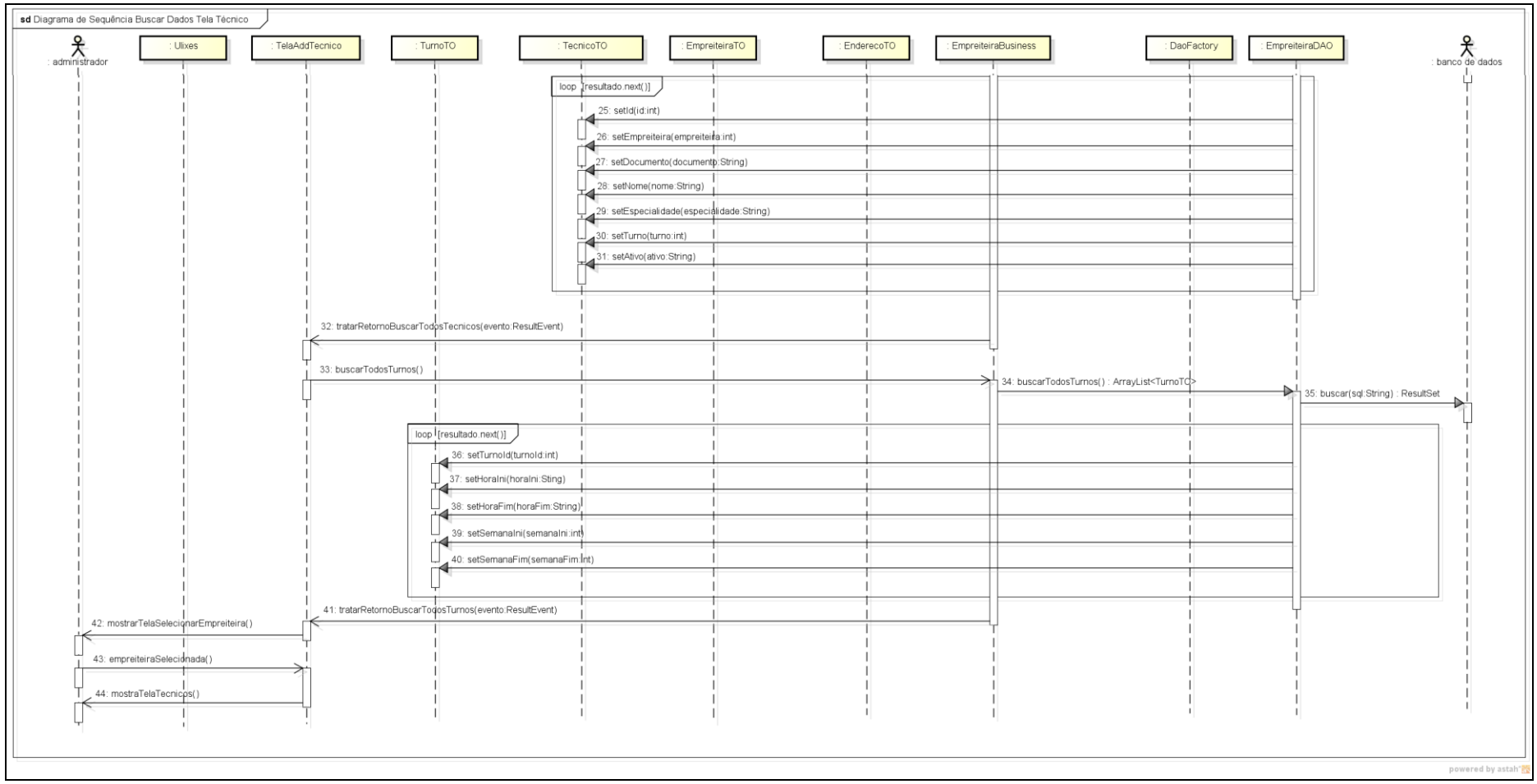


Figura 31 - Parte 2: Diagrama de Seqüência Buscar Dados Tela Técnico

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA CADASTRAR TÉCNICO

O caso de uso “Cadastrar Técnico” foi desenvolvido para permitir que usuários com perfil de administrador do sistema pudessem cadastrar novos. Um requisito necessário a este caso de uso é inserção previa das empreiteiras que serão vinculadas aos técnicos.

Estando na tela de gerência de técnicos, ao se clicar no botão ‘Inserir Técnico’ um novo painel é mostrado na parte inferior da tela, requisitando o nome, o telefone, o documento e o turno do novo técnico (Figura 32). Após inserir todas essas informações, o usuário deve clicar no botão ‘Inserir’. Se não ocorrer nenhum problema de validação, o novo técnico é inserido na base de dados.

Como se pode observar na Figura 33, após um novo técnico ser inserido, a instância da classe de interface com o usuário ‘TelaAddTécnico’ é atualizada .

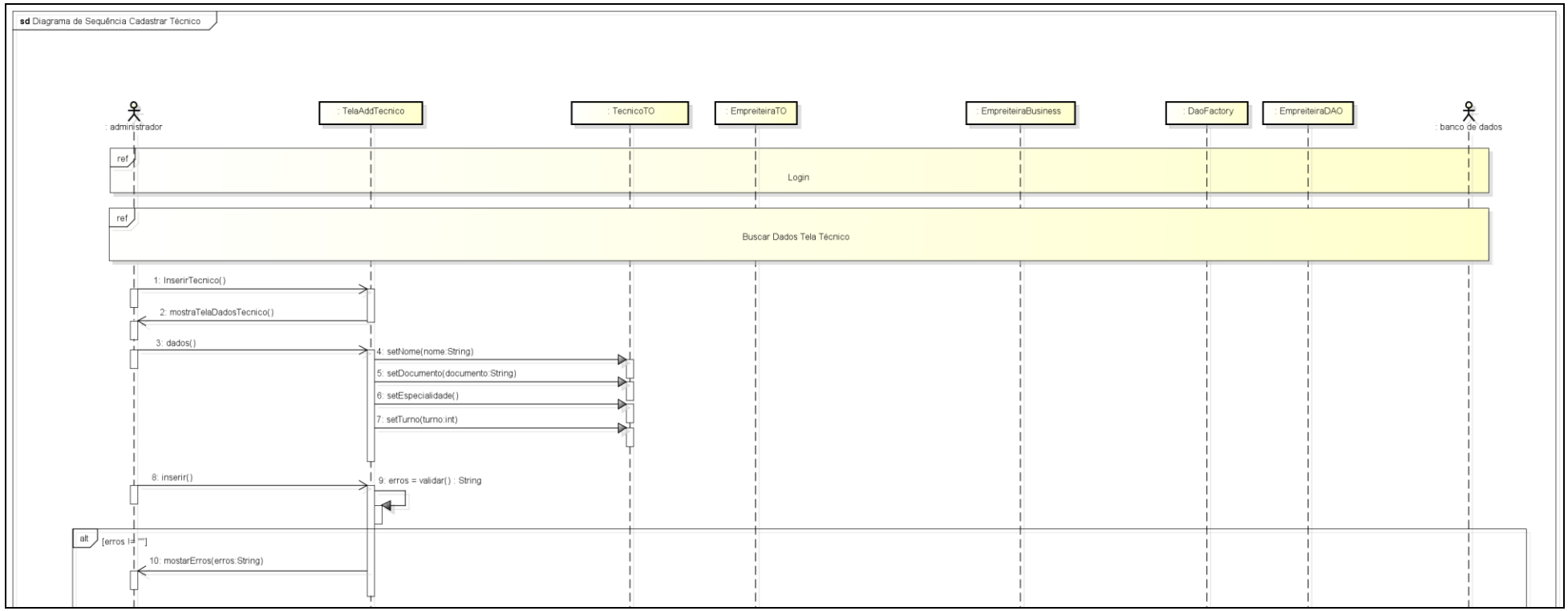


Figura 32 - Diagrama de Seqüência Cadastrar Técnico Parte 1

Fonte: Autoria própria.

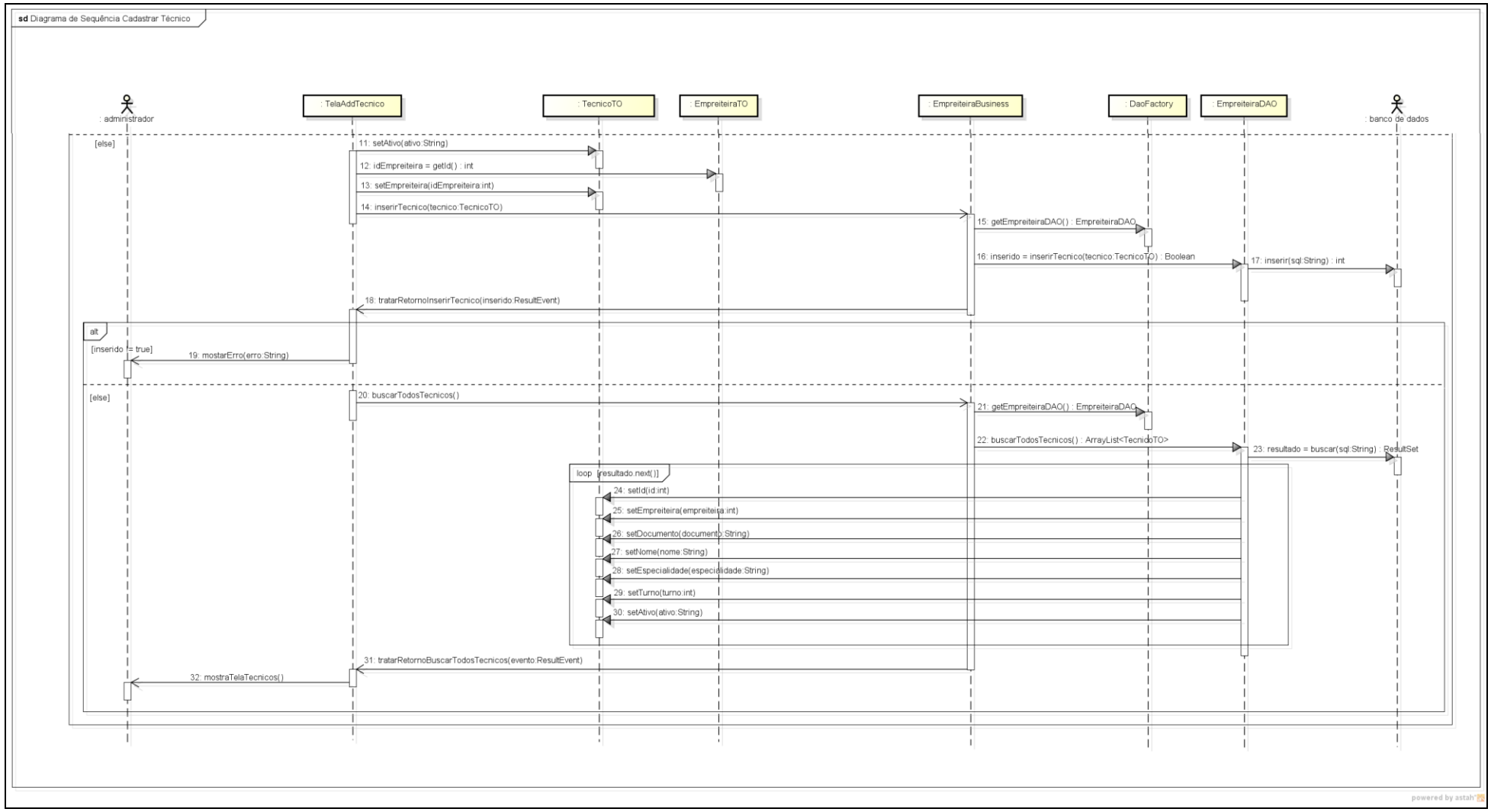


Figura 33 - Diagrama de Seqüência Cadastrar Técnico Parte 2

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA ALTERAR TÉCNICO

A Figura 34 e a Figura 35 ilustram como o processo de atualização de informações relativas a um técnico é manipulado pelo sistema.

Primeiramente, o usuário deve clicar duas vezes sobre o técnico que se deseja atualizar na tabela ‘Técnicos’. Feito isso, um novo painel será mostrado ao usuário na parte inferior da tela, com todas as informações do usuário acessíveis a edição.

Após efetuar as atualizações necessárias, o usuário deve clicar no botão ‘Atualizar’ para persistir tais informações no banco de dados. Um aspecto importante a se ressaltar é que as mesmas validações aplicadas ao processo de inserção de técnicos são utilizadas para o processo de atualização, impedindo a inserção de dados incorretos.

Além disso, nesse mesmo painel de alteração de dados do técnico está disponível um *checkbox* chamado ‘Ativo’, tirando-se a seleção deste, e persistindo as alterações no banco de dados, o técnico é removido do processo de envio de *tickets*.

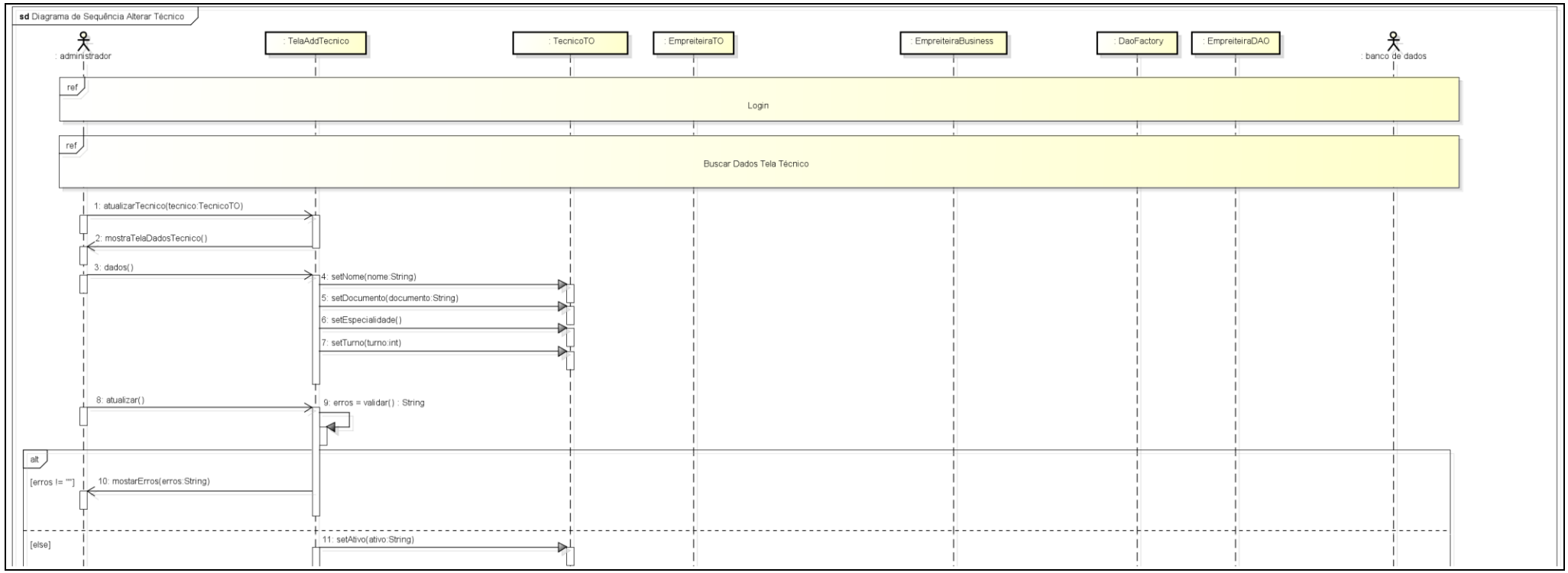


Figura 34 - Diagrama de Seqüência Alterar Técnico Parte 1

Fonte: Autoria própria.

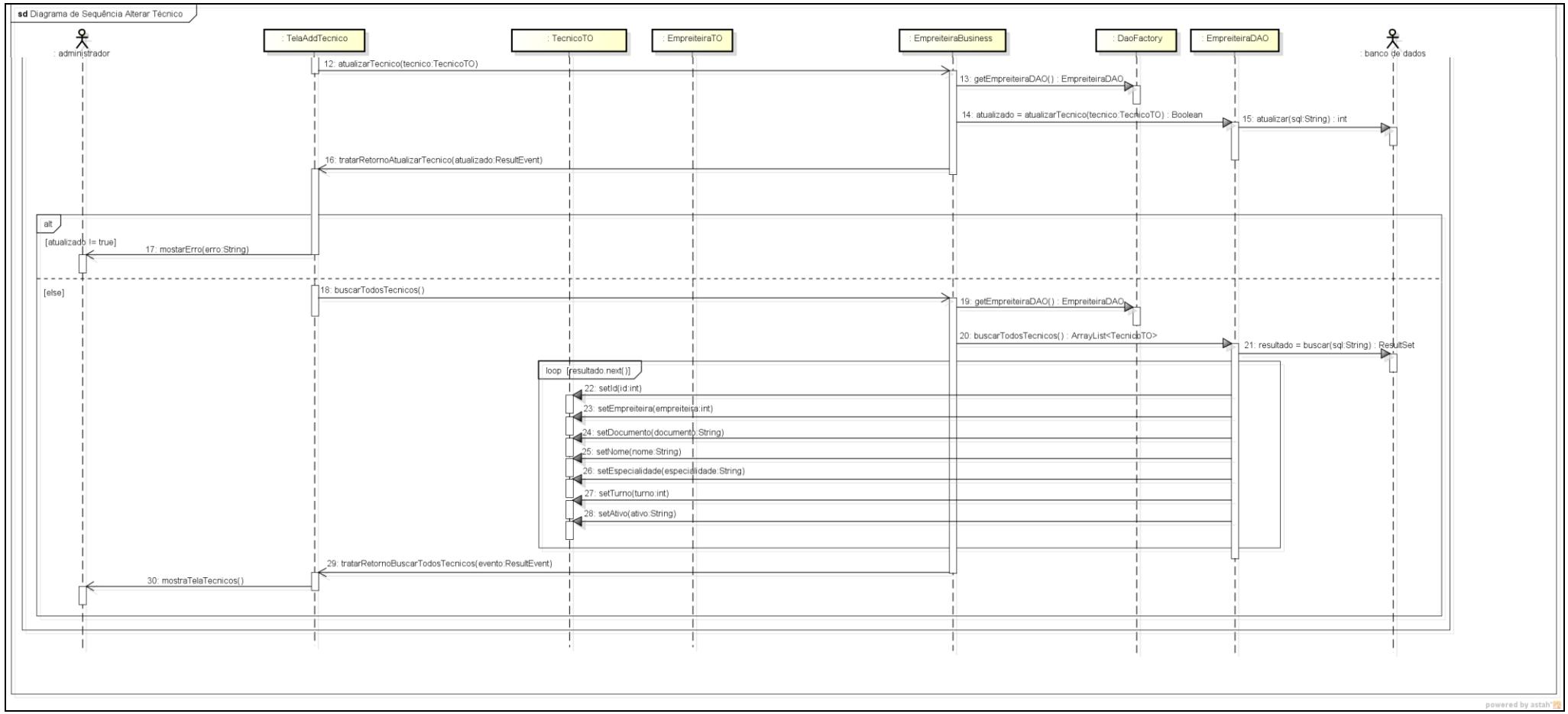


Figura 35 - Diagrama de Seqüência Alterar Técnico Parte 2

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA BUSCAR DADOS TELA PERFIL

O subcaso de uso “Buscar Dados Tela Perfil” tem como função pesquisar todos os perfis e funcionalidades disponíveis na base de dados e encaminhá-los através de todas as camadas do sistema até a interface com o usuário.

A Figura 36 demonstra como este processo é realizado. Ao se iniciar a tela de gerência de perfis, o método “buscarPerfis” é chamado no objeto da classe “UsuarioBusiness”. Este, por sua vez, chama o método “buscarTodosPerfis” em uma instância da classe abstrata ‘UsuarioDAO’, e este método popula objetos da classe “ItemMenu” com os dados relativos aos perfis constantes na base de dados.

Logo após tais dados serem obtidos, a instância da classe “UsuarioBusiness” ordena ao objeto da classe ‘UsuarioDAO’ que busque todos as funcionalidades presentes no sistema. Mais uma vez, objetos da classe ‘ItemMenu’ são preenchidos com dados relativos as funcionalidades do sistema e devolvidos a interface.

Por fim, a tela de gerência de perfis é mostrada ao usuário. Um aspecto importante a se ressaltar é que esta tela pode ser utilizada tanto pelo processo de inserção de perfis quanto pelo processo de atualização destes.

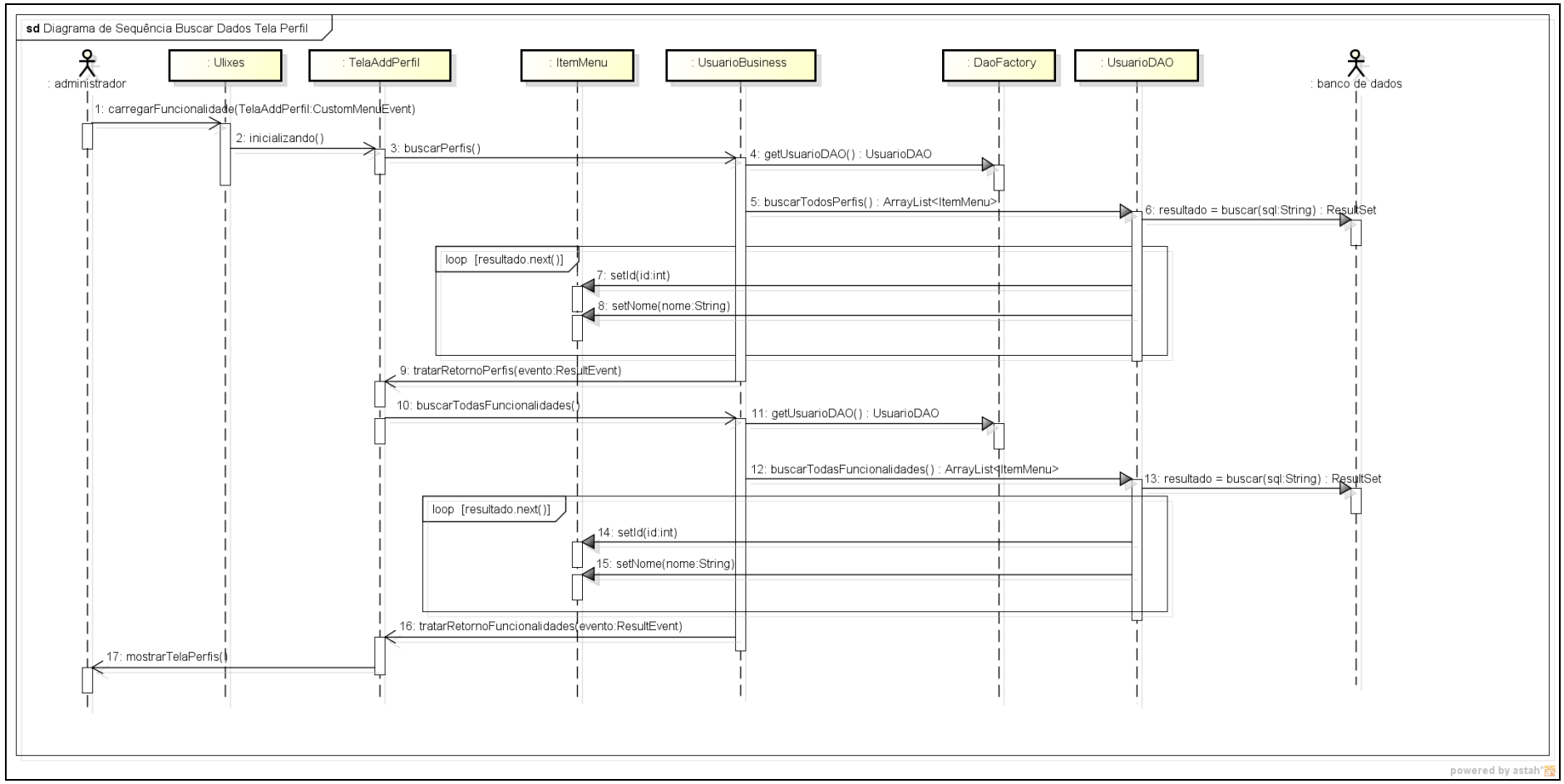


Figura 36 - Diagrama de Seqüência Buscar Dados Tela Perfil

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA CADASTRAR PERFIL

O processo de cadastro de perfis é muito importante visto que por ele é possível a criação de diferentes níveis de acesso as funcionalidades do *ulixes*, ficando a cargo do administrador do sistema definir quais níveis de acesso serão estes.

Como se pode observar na Figura 37, o primeiro passo para a inserção de um novo perfil é o clique sobre o botão inserir na parte inferior da página de gerência de perfis. Após tal procedimento, um novo painel é apresentado na parte inferior da página, com o título "Adicionar Perfil". Este painel requisita apenas um parâmetro, o nome do novo perfil, sendo que é necessário após o preenchimento deste que o usuário clique no botão "Inserir Funcionalidades" para que sejam selecionadas as telas as quais este perfil terá acesso.

No novo painel mostrado ao usuário após o último procedimento, existem campos de checagem vinculados a descrição da funcionalidade, que podem ser selecionadas com um simples clique sobre o *checkbox.*

Por fim, o usuário deve clicar no botão "inserir" para que o novo perfil seja persistido na base de dados. Para tal, é utilizada um instância da classe abstrata "UsuarioDAO", que obtém os novos dados criados pelo usuário de um objeto de transferência "PerfilTO". Logo após os dados serem persistidos, a interface com o usuário é atualizada com o novo perfil (Figura 38).

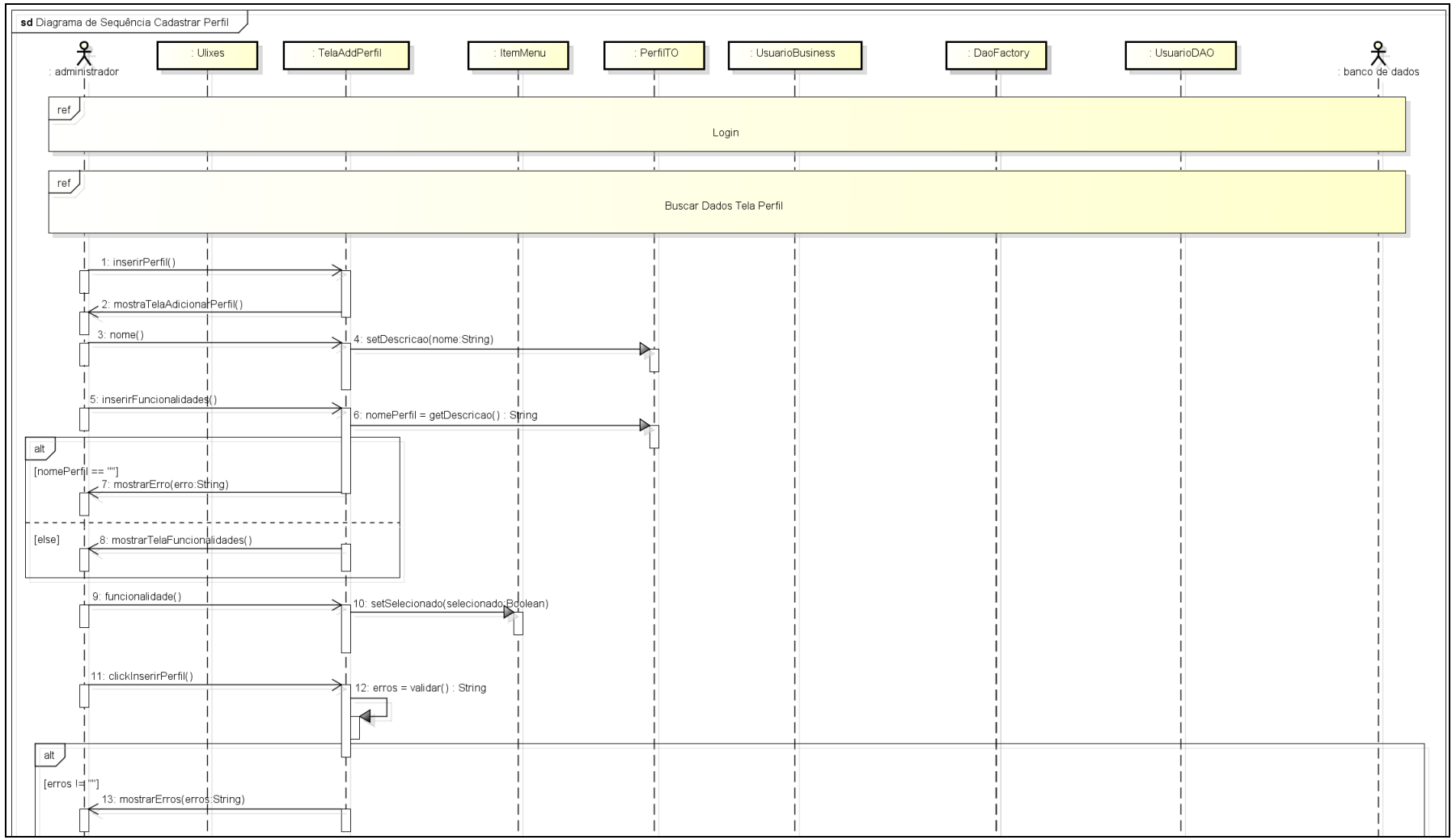


Figura 37 - Parte 1: Diagrama de Seqüência Cadastrar Perfil

Fonte: Autoria própria.

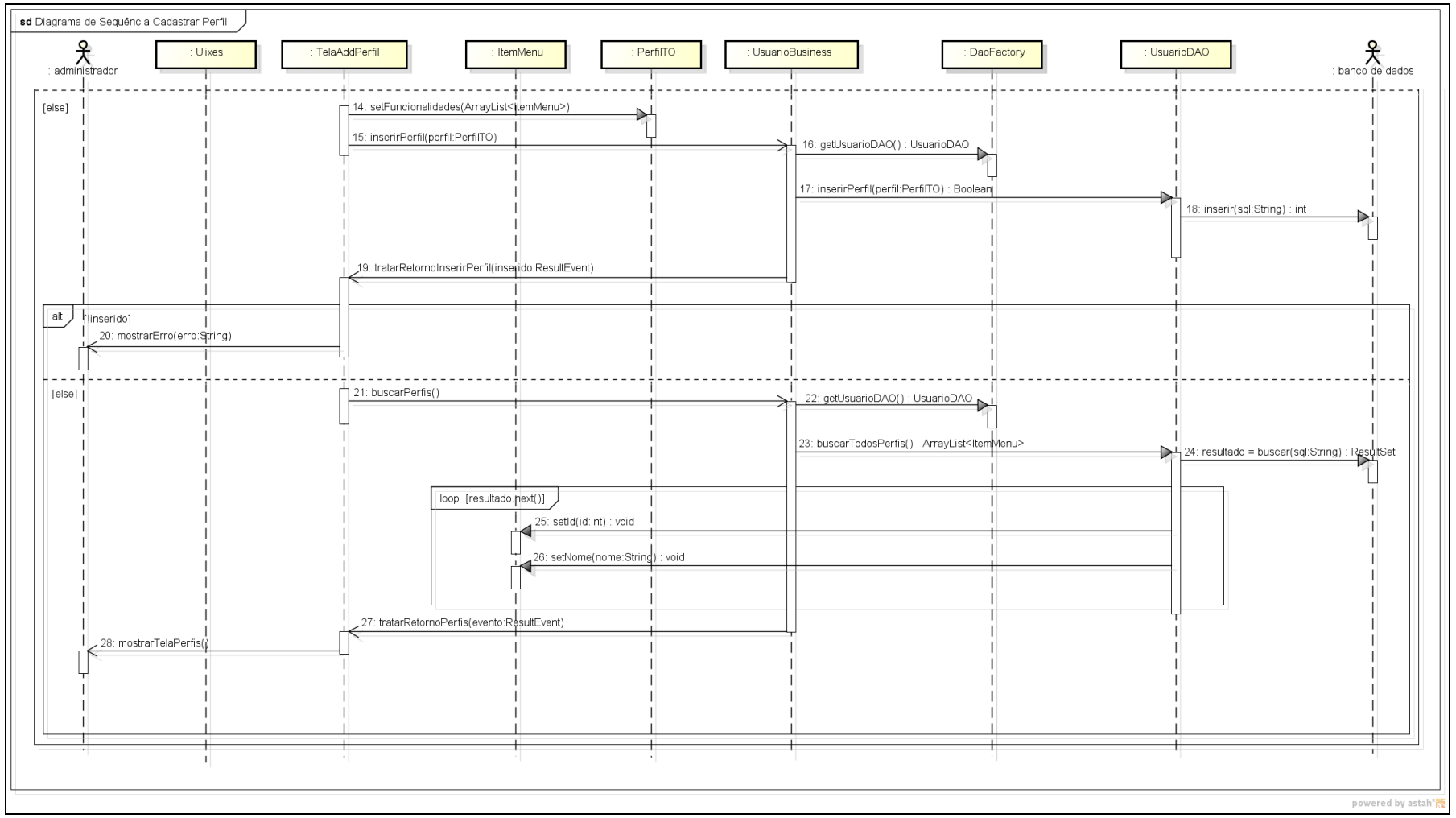


Figura 38 - Parte 2: Diagrama de Seqüência Cadastrar Perfil

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA ALTERAR PERFIL

A atualização de perfis é um processo muito simples no sistema Ulixes. Para que o painel de edição de perfis seja habilitado basta ao usuário um clique simples sobre o nome do perfil que se deseja modificar (Figura 39).

Após esse primeiro passo, um painel é mostrado ao usuário, com todas as funcionalidades disponíveis no sistema e também apresentando as funcionalidades que foram selecionadas previamente para este perfil. Neste tela, pode-se tanto adicionar novas funcionalidades quanto remover as que previamente foram atreladas a tal perfil.

O último passo necessário ao processo de atualização de perfis é um clique sobre o botão “Atualizar”, que persistirá as novas funcionalidades selecionadas para esse perfil no banco de dados (Figura 40).

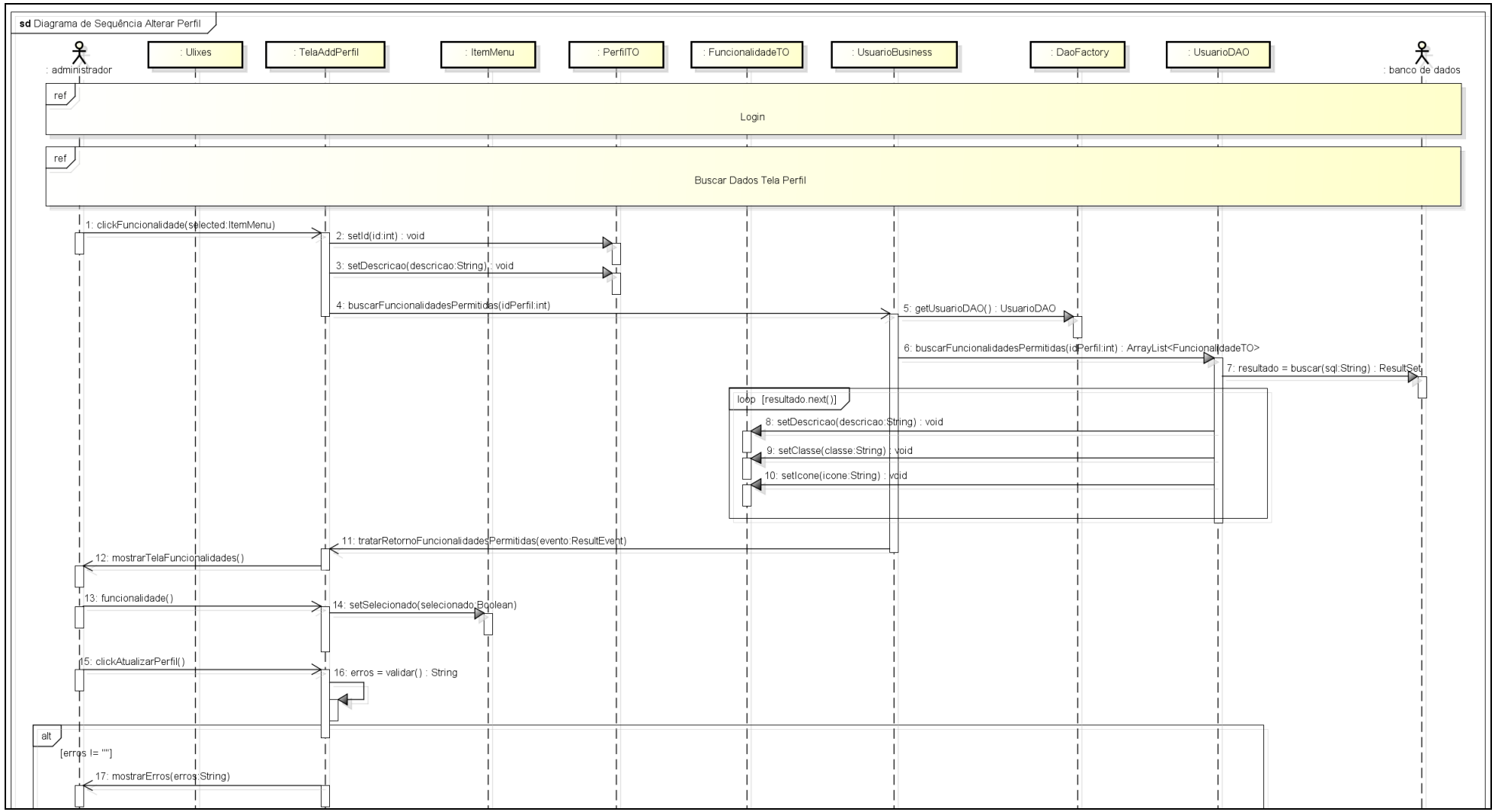


Figura 39 - Parte 1: Diagrama de Seqüência Alterar Perfil

Fonte: Autoria própria.

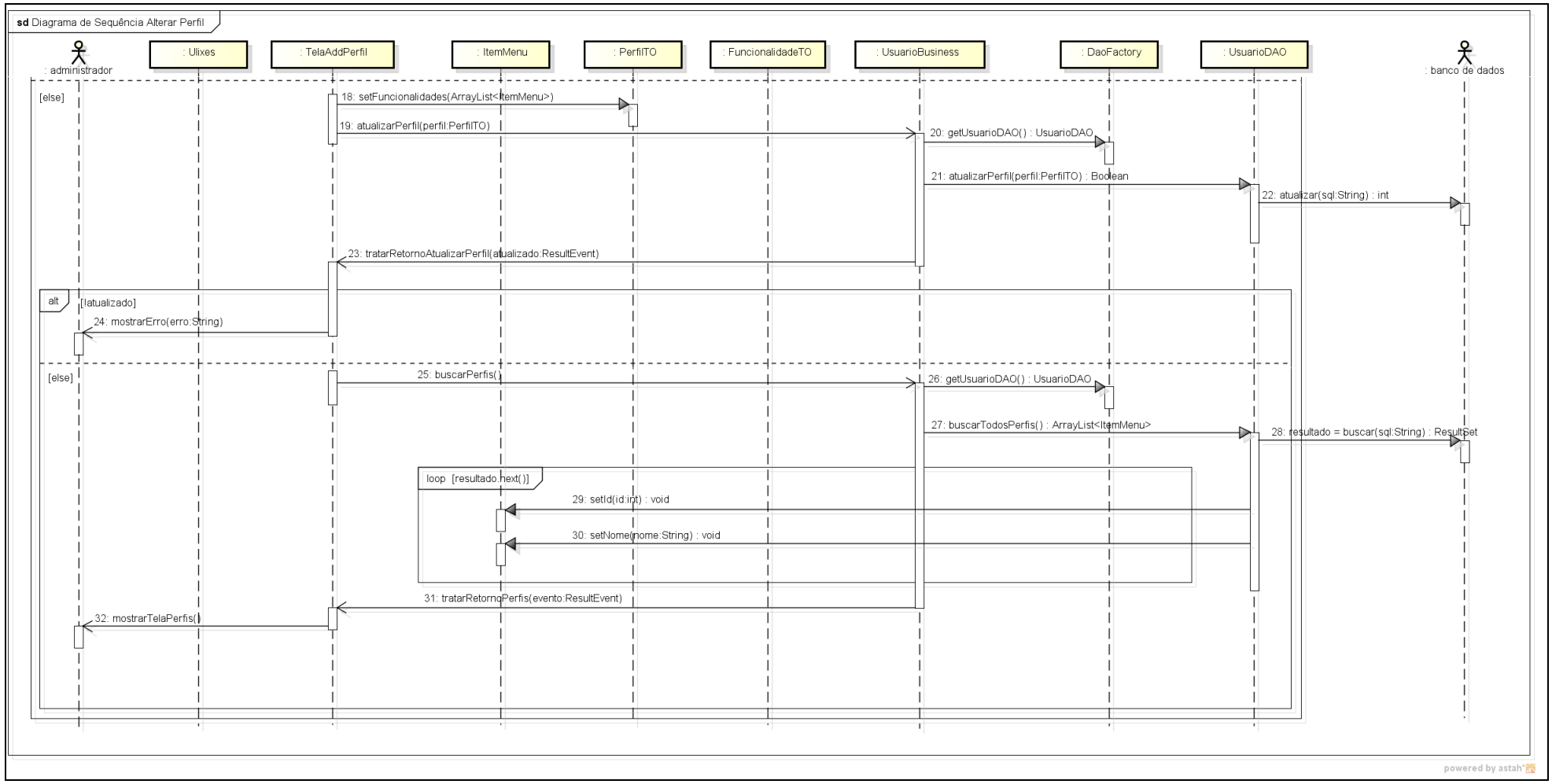


Figura 40 - Parte 2: Diagrama de Seqüência Alterar Perfil

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA BUSCAR DADOS TELA USUÁRIO

O subcaso de uso “Buscar Dados Tela Usuário” está inserido nos casos de uso “Cadastrar Usuário” e “Alterar Usuário”, e tem com função principal buscar todas as informações relativas aos usuários do sistema para a tela de interface com o usuário “TelaAddUsuario”.

Pode-se observar conforme a Figura 41 que logo após a tela de gerência de usuários ser inicializada pelo sistema é ordenado ao objeto da classe “UsuarioBusiness” a busca de todos os perfis cadastrados no sistema. Este objeto, por sua vez, chama o método “buscarTodosPerfis” da instância da classe abstrata “UsuarioDAO”. Esta classe faz uma consulta contra a base de dados para obter tais informações, salvado-as em objetos “ItemMenu” que são retornados até a interface com o usuário.

Após tais dados serem retornados, uma nova chamada é efetuada à classe “UsuarioBusiness”, ordenando a busca dos dados relativos aos usuários cadastrados na base de dados. Novamente está classe faz uma chamada a instância da classe abstrata “UsuarioDAO”, que consulta a base de dados para obter tais informações e popular objetos da classe “UsuarioTO” que são devolvidos à interface.

Como último passo desse sub caso de uso, a tela gerência de usuários é apresentada ao usuário.

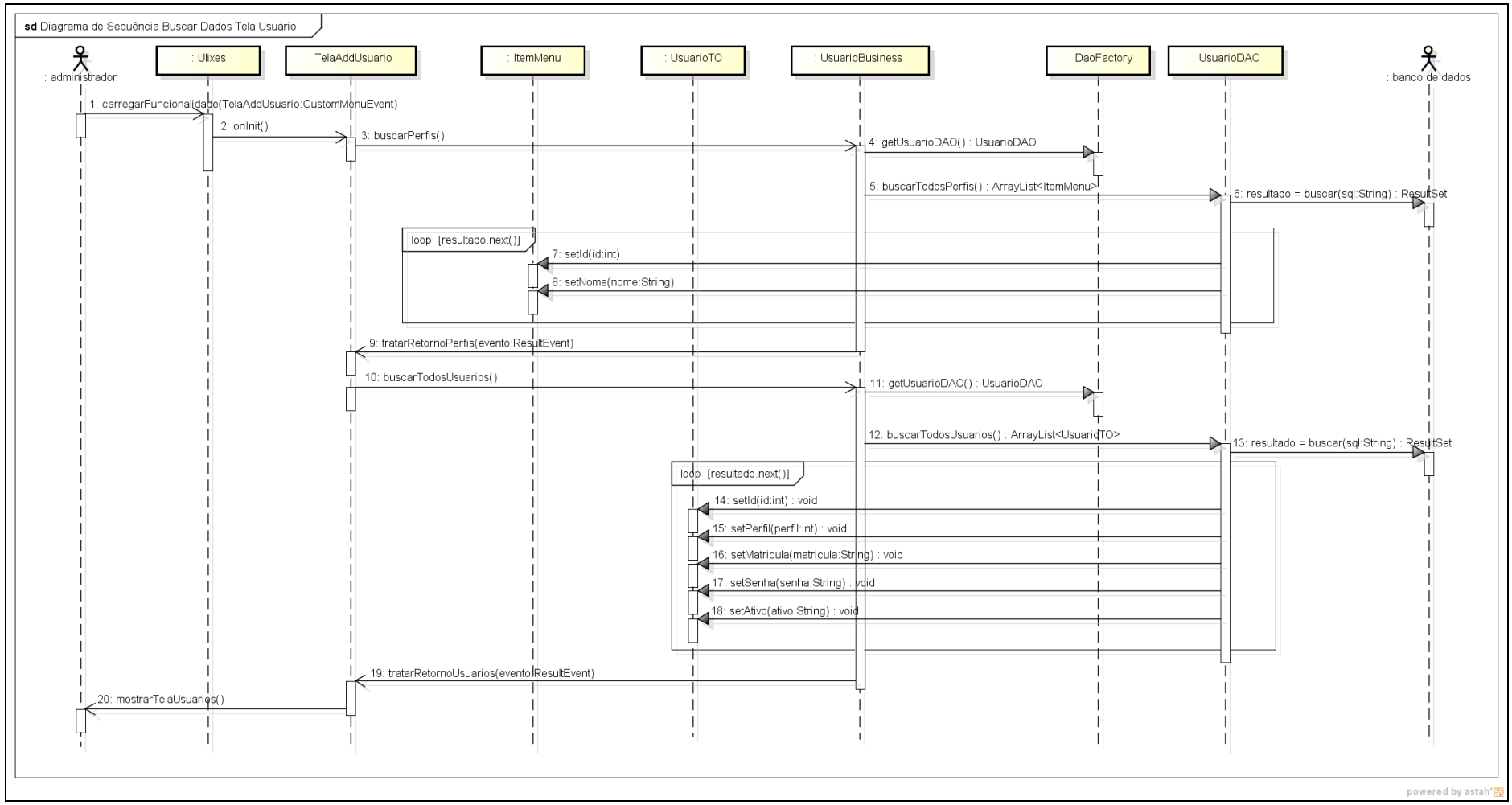


Figura 41 - Diagrama de Seqüência Buscar Dados Tela Usuário

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA CADASTRAR USUÁRIO

Na Figura 42 e na Figura 43 é apresentado o diagrama de seqüência “Cadastrar Usuário”, que é responsável pela inserção de novos usuários no sistema.

O primeiro passo para a inserção de um novo usuário é abrir o *popup* responsável por esse processo. Isso pode ser feito clicando-se no botão “Inserir Usuário” da tela de gerência de usuários.

No *popup* intitulado “Insira os Dados do Novo Usuário” é requisitado ao usuário o preenchimento da matrícula, da senha e do perfil do novo usuário. Esses dados são validados ao se clicar no botão Inserir e se nada de anormal for encontrado os dados são persistidos na base de dados.

Após a inserção do novo usuário, a objeto da classe de interface com o usuário é atualizada com as novos dados recém inseridos.

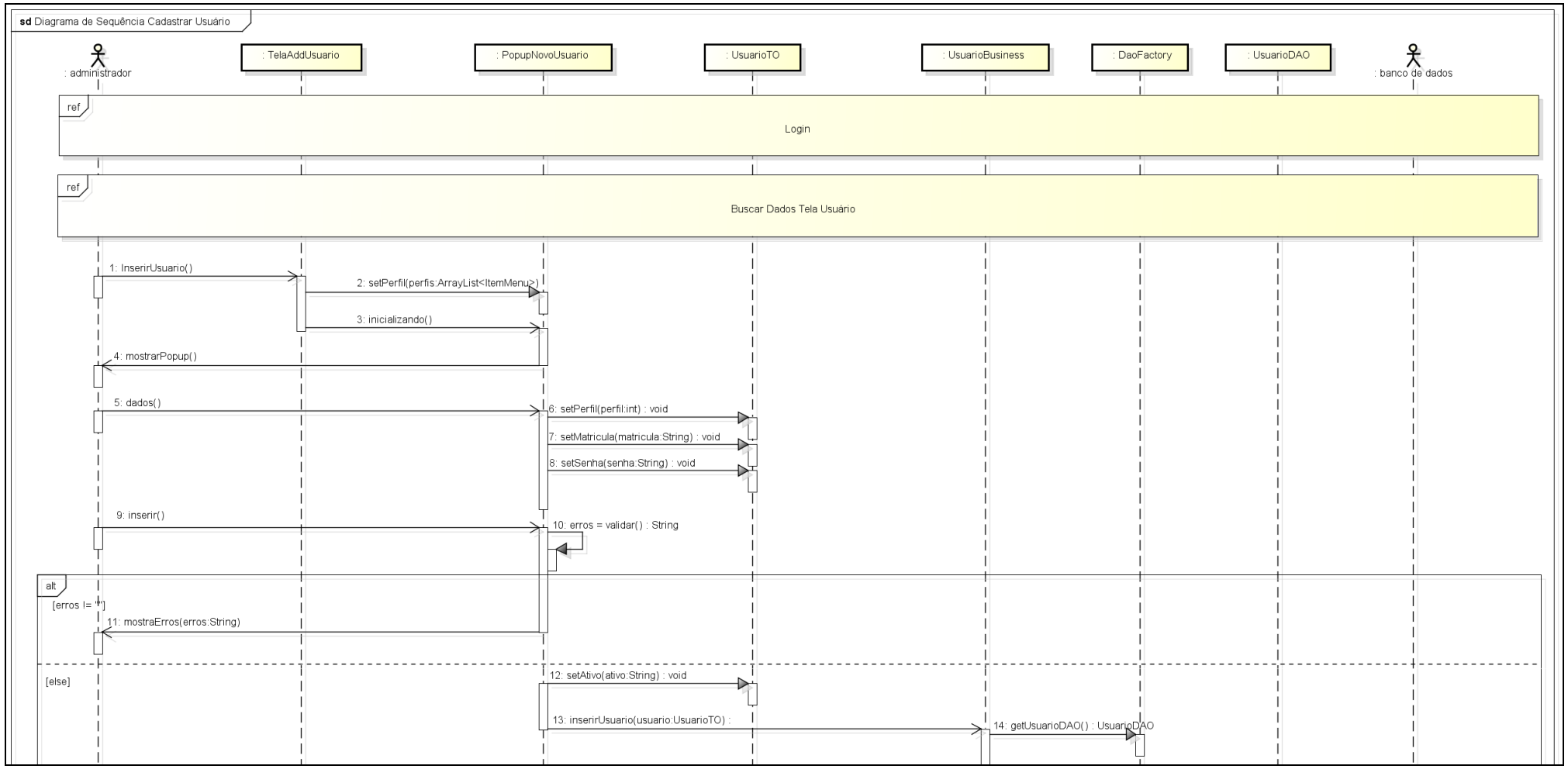


Figura 42 - Parte 1: Diagrama de Seqüência Cadastrar Usuário

Fonte: Autoria própria.

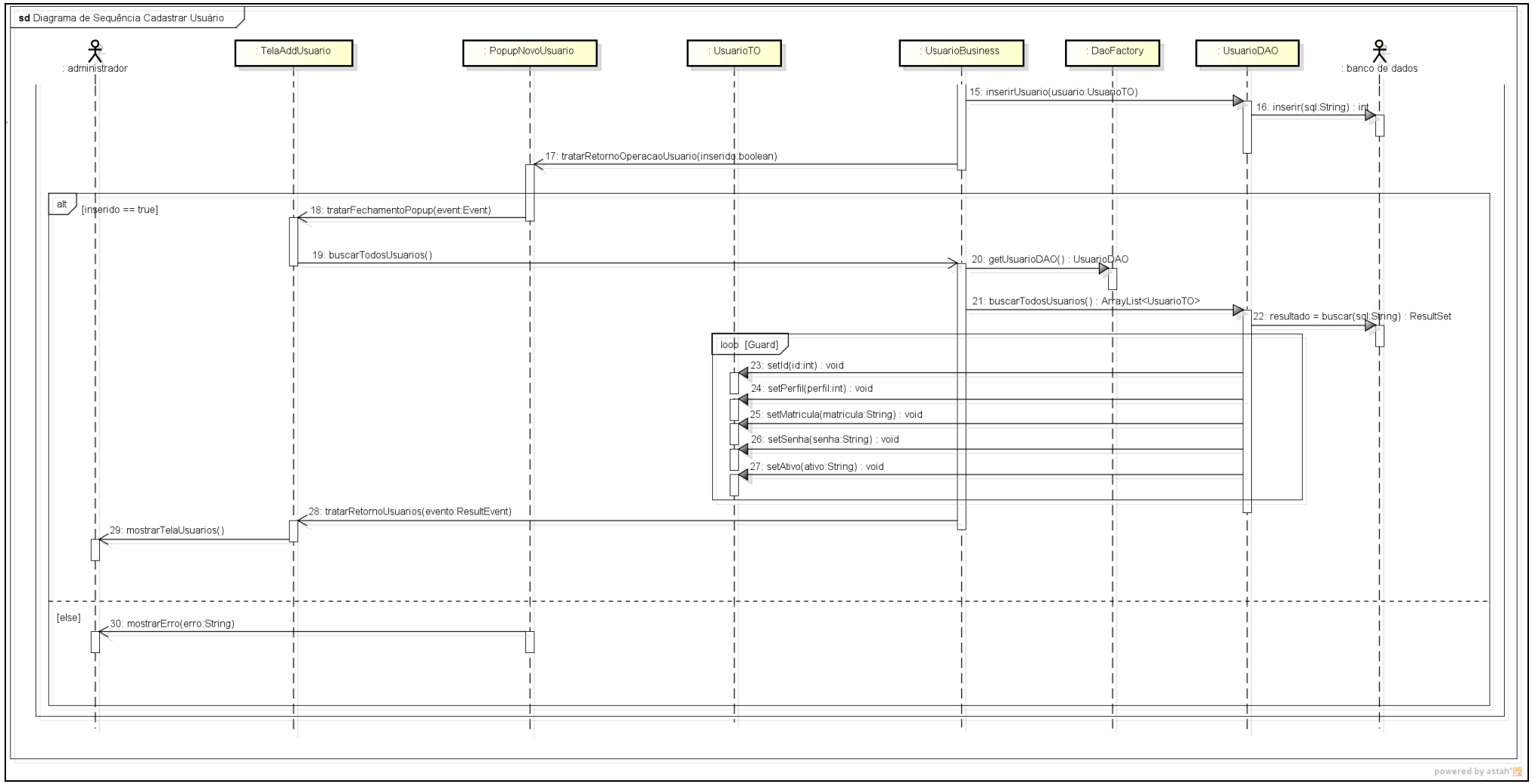


Figura 43 - Parte 2: Diagrama de Seqüência Cadastrar Usuário

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA ALTERAR USUÁRIO

O processo de atualização de usuários é bastante similar ao processo de atualização de empreiteiras. Para ter acesso a funcionalidade “Alterar Usuário”, basta um clique duplo sobre o nome do usuário que se deseja modificar na tela de gerência de usuários. Feito isso, um *popup* (Figura 44) será disponibilizado para que os dados cadastrais do usuário do sistema possam ser alterados.

Estando nesse *popup*, todos os dados cadastrais do usuário são atualizáveis, sendo que as mesmas validações de dados aplicadas ao processo de criação de usuários são aplicadas ao processo de atualização destes. Outro ponto importante a se destacar é que um usuário pode ter seu acesso ao sistema removido por esse processo. Para isso, é necessário desmarcar a opção “Ativo” do formulário apresentado.

Por fim, após os dados serem persistidos através de um clique sobre o botão “Atualizar”, a interface com o usuário é atualizada com os novos dados (Figura 45).

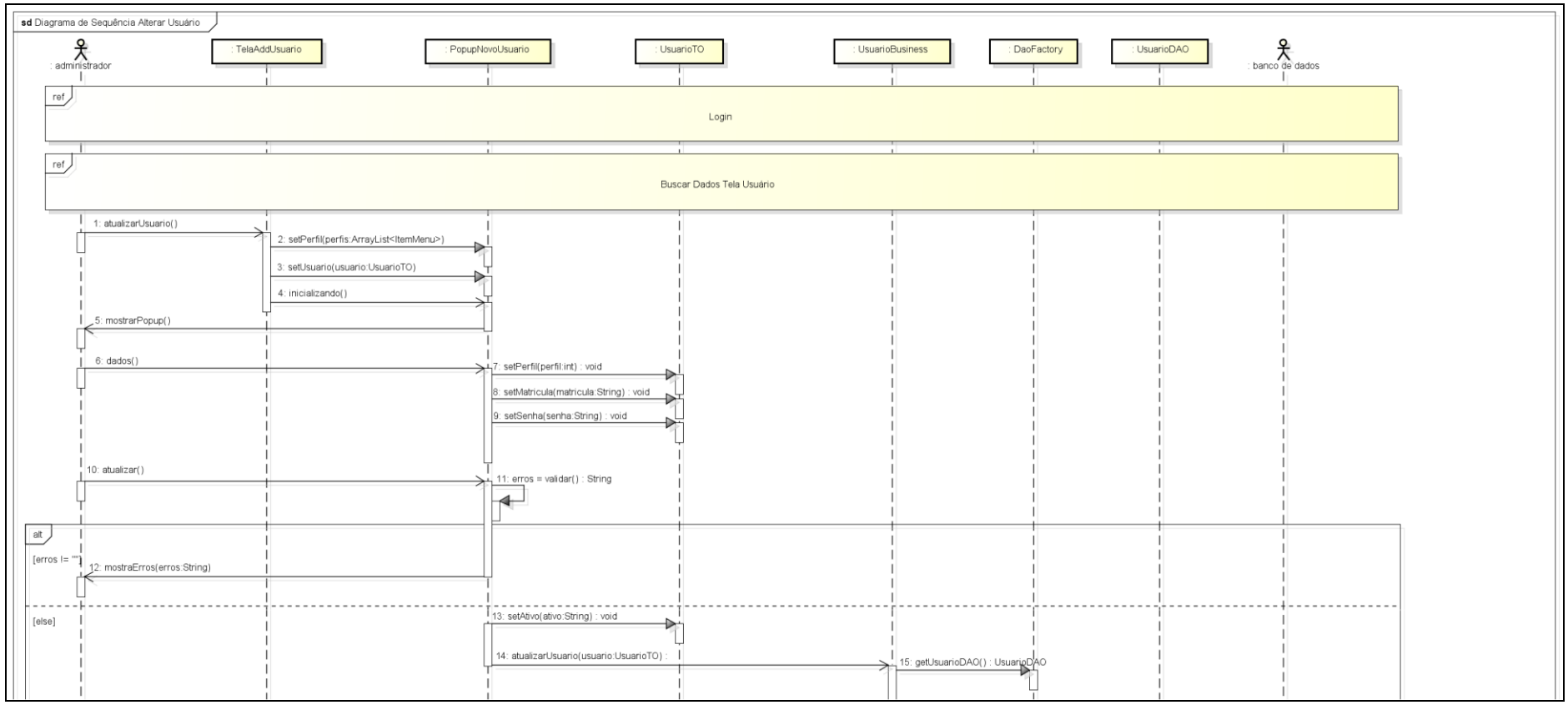


Figura 44 - Parte 1: Diagrama de Seqüência Alterar Usuário

Fonte: Autoria própria.

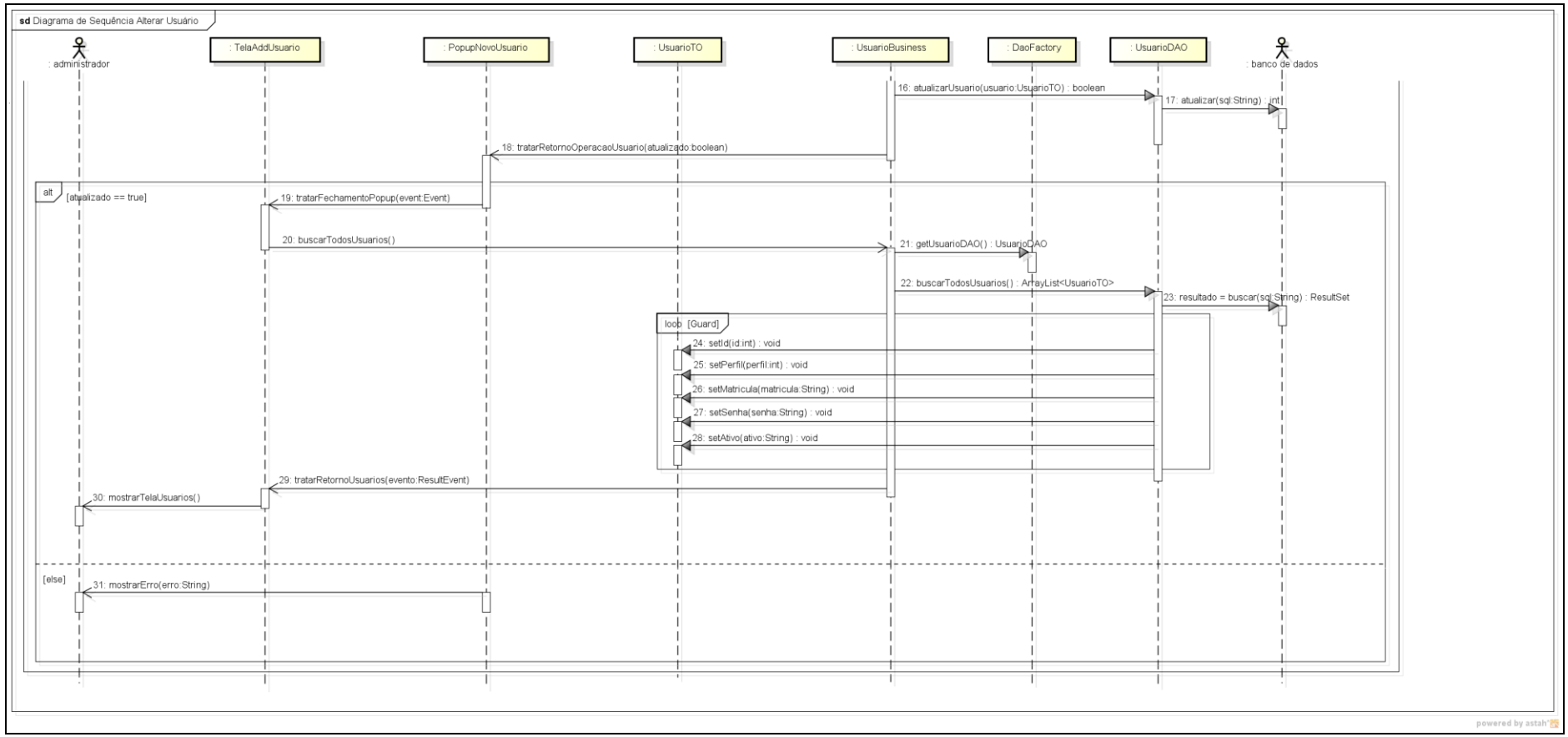


Figura 45 - Parte 2: Diagrama de Seqüência Alterar Usuário

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA DE LOGIN

A etapa de login é um subcaso de uso que participa de praticamente todos os outros casos de uso. Ela serve não apenas para fins de segurança, identificando os usuários com permissão de acesso ao sistema, mas também permite a customização da aplicação.

Conforme mostram a Figura 46 e a Figura 47, após a validação positiva do nome de usuário e senha, as funcionalidades permitidas para o indivíduo que acaba de ser autenticado – e que estão definidas em seu perfil – são consultadas na base de dados. O retorno desta consulta é utilizado para a montagem de um menu dinâmico. Esta estrutura possibilita que os conjuntos de ferramentas visualizados por cada usuário sejam tão distintos quanto for necessário.

Como o produto de um projeto Adobe Flex, em termos de interface, é um arquivo SWF (*Shockwave Flash*) encapsulado em uma página html simples, não é possível copiar a url do navegador de um usuário já logado ao Ulixes e acessar as mesmas telas que ele está vendo, ao colá-la em um *browser* diferente. Isso dispensa o uso de *servlets* do tipo *Filter*, comuns em projetos Java para a web.

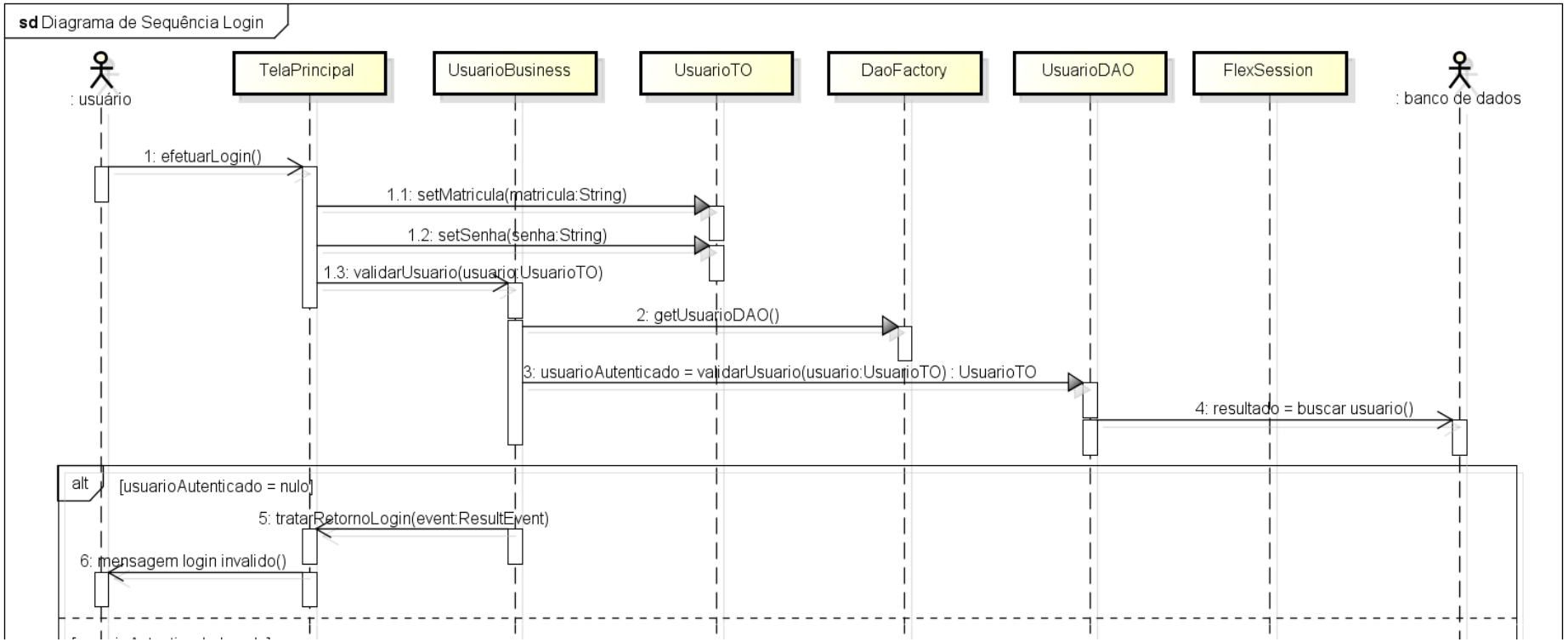


Figura 46 - Parte 1: Diagrama de Seqüência Login

Fonte: Autoria própria.

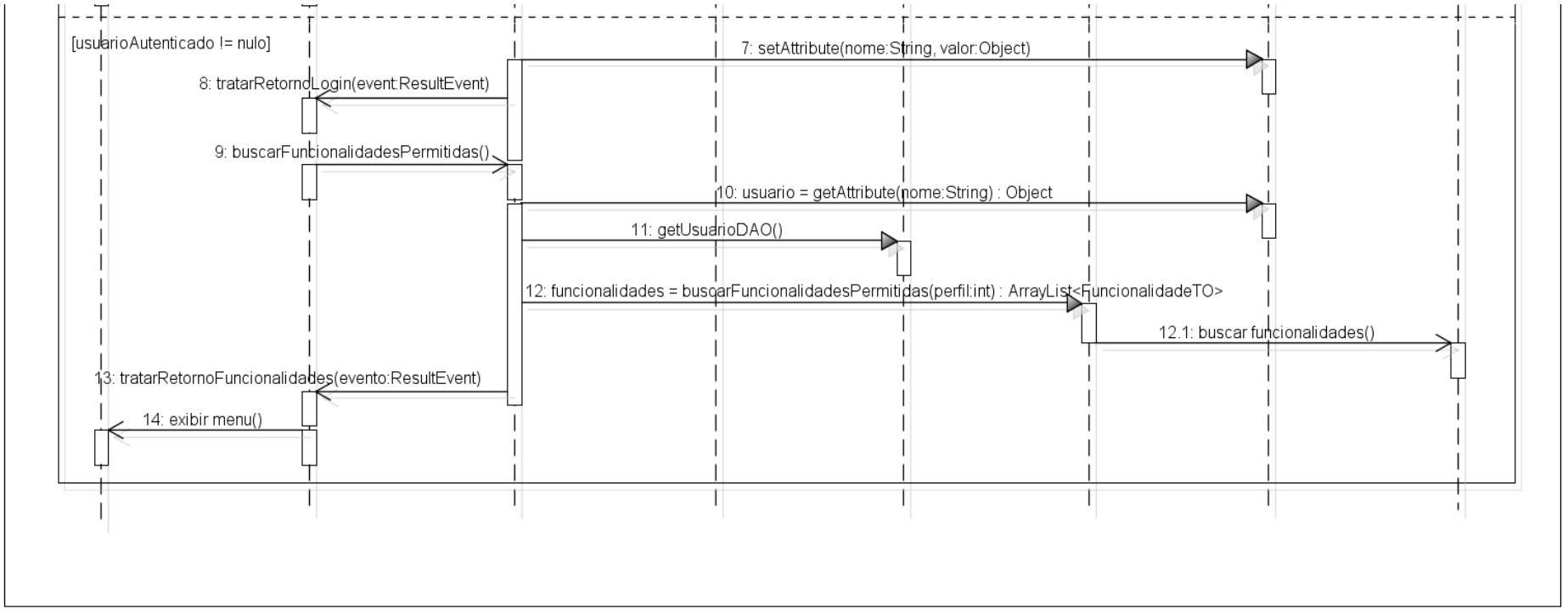


Figura 47 - Parte 2: Diagrama de Seqüência Login

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA CADASTRAR CLIENTE

O subcaso de uso “Cadastrar Cliente” é uma parte do caso de uso mais amplo “Inserir Ticket”. Não se pretende manter no Ulixes dados de clientes que não tenham sido atendidos ou que não serão atendidos num futuro imediato, por questões de otimização. Partindo deste pressuposto, ao invés de disponibilizar uma tela separada para o cadastro de novos clientes, esta tarefa é feita durante a criação do ticket.

A Figura 48, a Figura 49 e a Figura 50 ilustram esse processo de adição de clientes à base. O preenchimento dos dados é feito em fases, sendo primeiramente fornecidos os dados diretos do cliente – cpf/cnpj, telefones de contato, etc. – e depois os dados do endereço. Para evitar erros de digitação e variações de grafia, o usuário precisa apenas preencher o CEP, e o sistema busca em uma tabela interna as demais informações (cidade, bairro, rua). Como cidades menores podem utilizar um único CEP para designar várias ruas ou bairros, se a busca retornar mais de uma possibilidade o usuário é solicitado a escolher dentre as que estão disponíveis, em uma janela *popup*.

Por fim, o salvamento das informações referentes ao cliente é feita no ato de inserção do ticket, após as validações necessárias para garantir que nenhum dado inválido seja aceito.

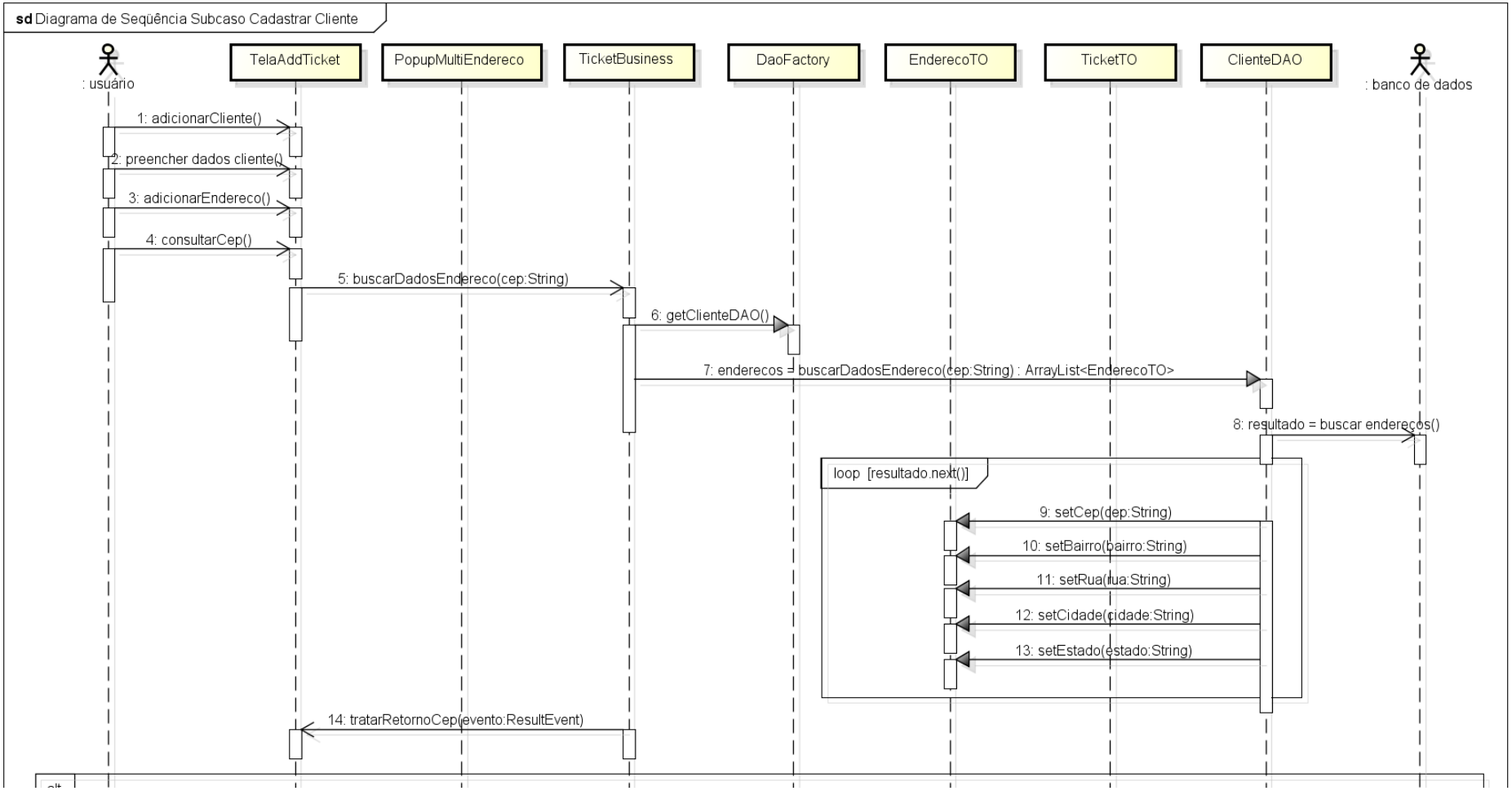


Figura 48 - Parte 1: Diagrama de Seqüência Cadastrar Cliente

Fonte: Autoria própria.

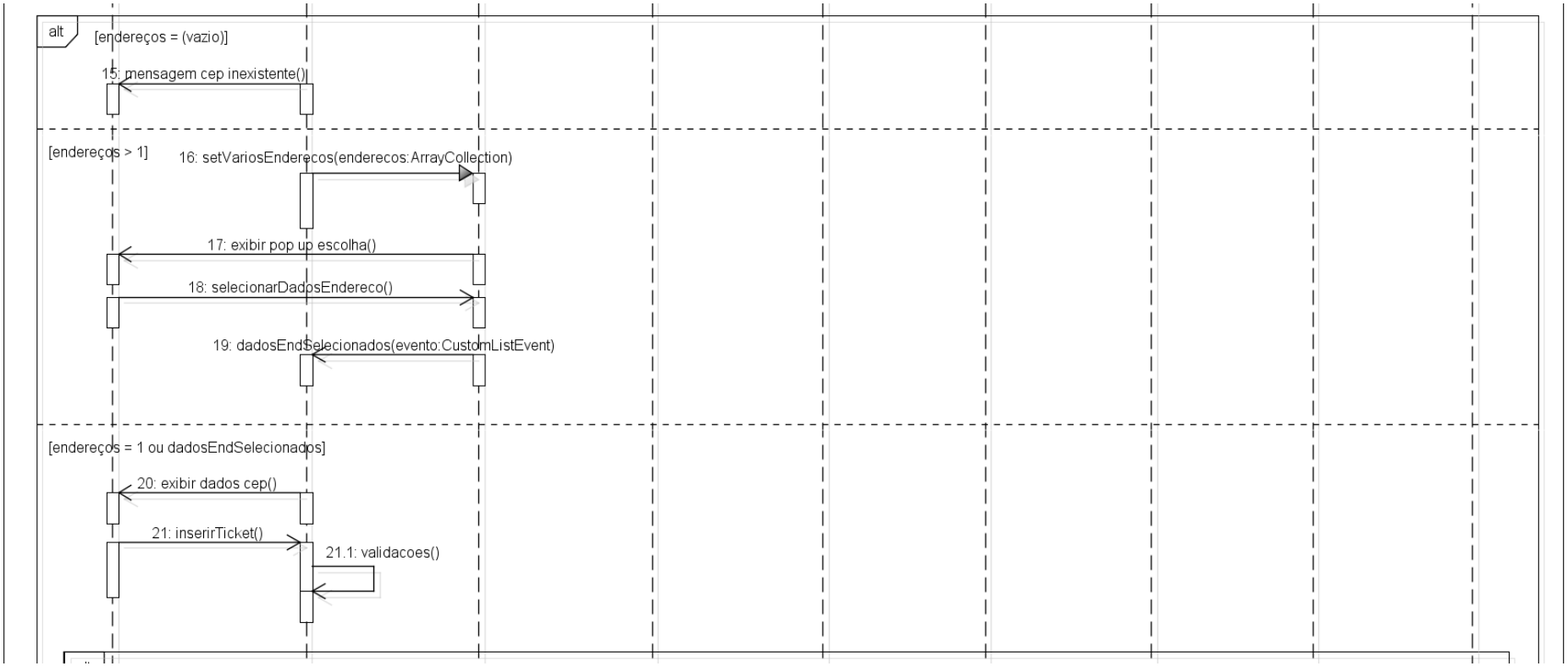


Figura 49 - Parte 2: Diagrama de Seqüência Cadastrar Cliente

Fonte: Autoria própria.

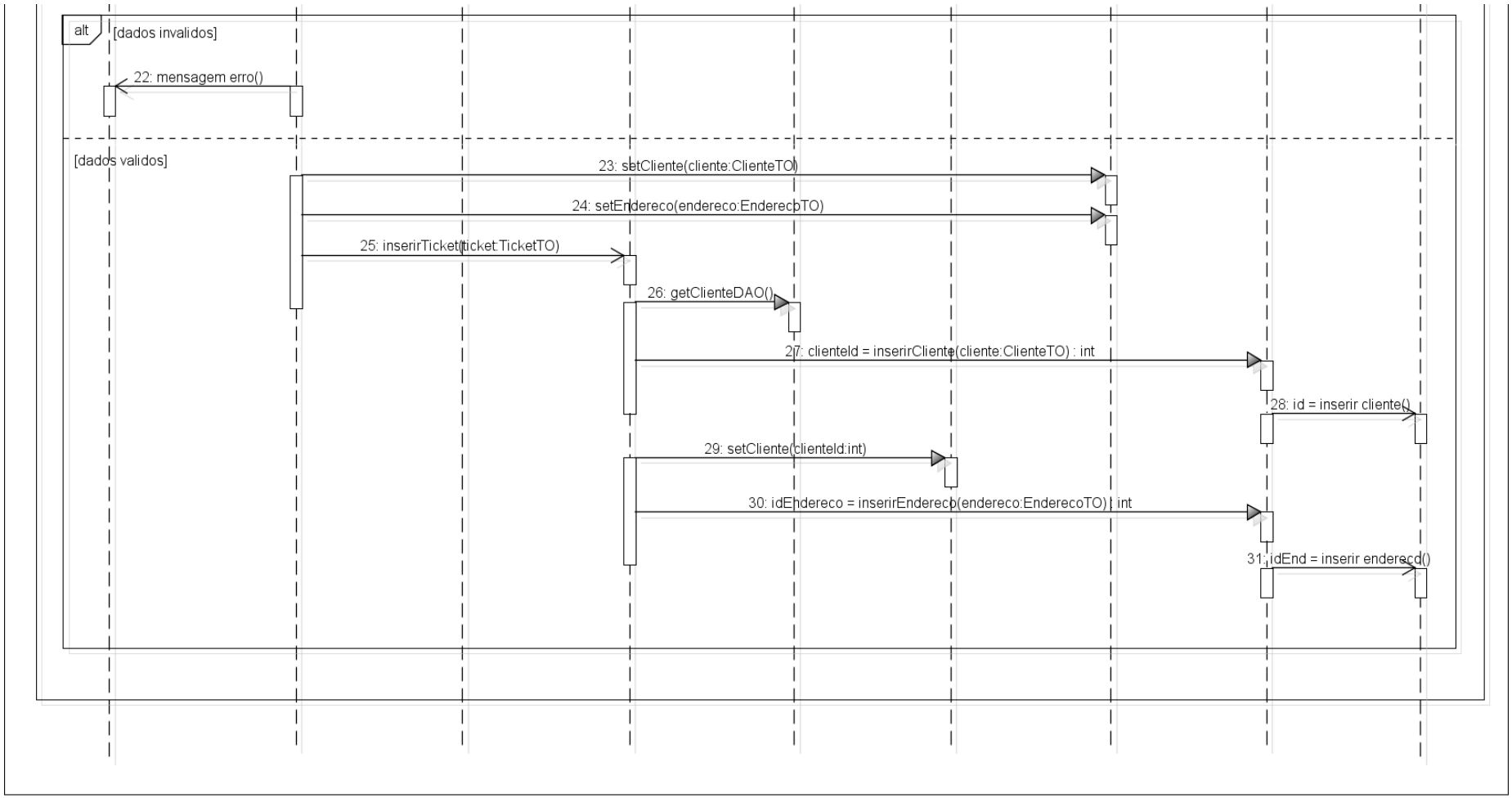


Figura 50 - Parte 3: Diagrama de Seqüência Cadastrar Cliente

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA CONSULTAR CLIENTE

Assim como no tópico anterior, o subcaso de uso “Consultar Cliente” é um auxiliar do caso “Inserir Ticket”. Adicionalmente, também participa do caso de uso “Alterar Cliente”. As telas foram modularizadas de modo a obter um reaproveitamento de código, e o mecanismo de busca do cliente descrito na Figura 51 e na Figura 52 compreende um módulo.

A busca por um cliente já existente na base é feita com o documento do mesmo, sendo que documento pode ser o cpf (no caso de pessoa física) ou cnpj (no caso de pessoa jurídica). Para cada documento, existe uma única conta cliente; endereços múltiplos são solucionados em um relacionamento 1:N com a entidade cliente cadastrada.

Por essa razão, para um cliente podem retornar um ou mais endereços. Este detalhe permite, inclusive, que em casos de mudança de residência mantenha-se um histórico dos locais anteriores nos quais o cliente já foi atendido.

O resultado da busca consiste nos dados do cliente, e em uma lista de endereços resumidos. Clicando-se em um item da lista, pode-se ver mais detalhes sobre o mesmo; esta interação não aparece explícita no diagrama de seqüência por ser controlada inteiramente por métodos privados dos componentes TelaAddTicket e TelaAltCliente.

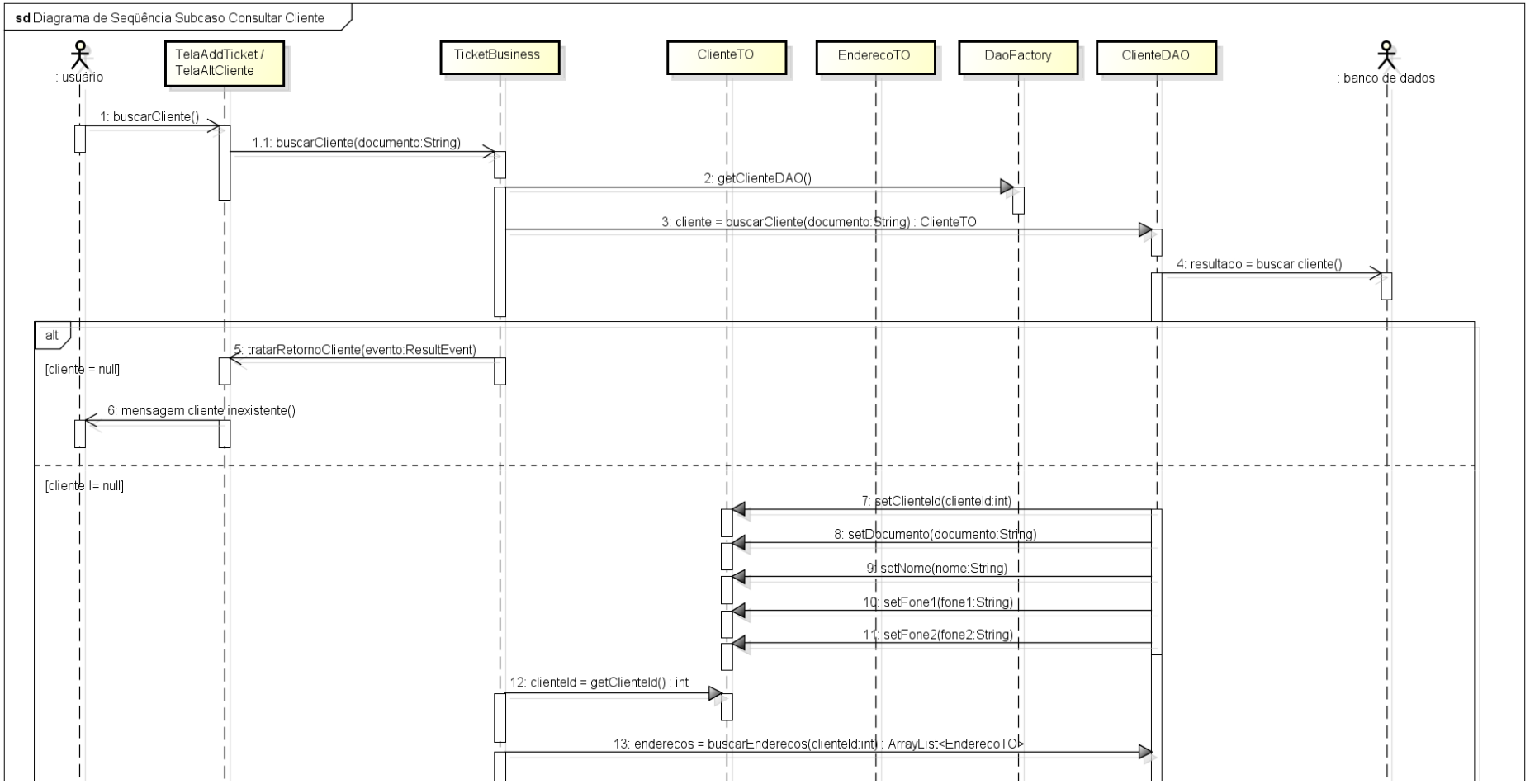


Figura 51 - Parte 1: Diagrama de Seqüência Consultar Cliente

Fonte: Autoria própria.

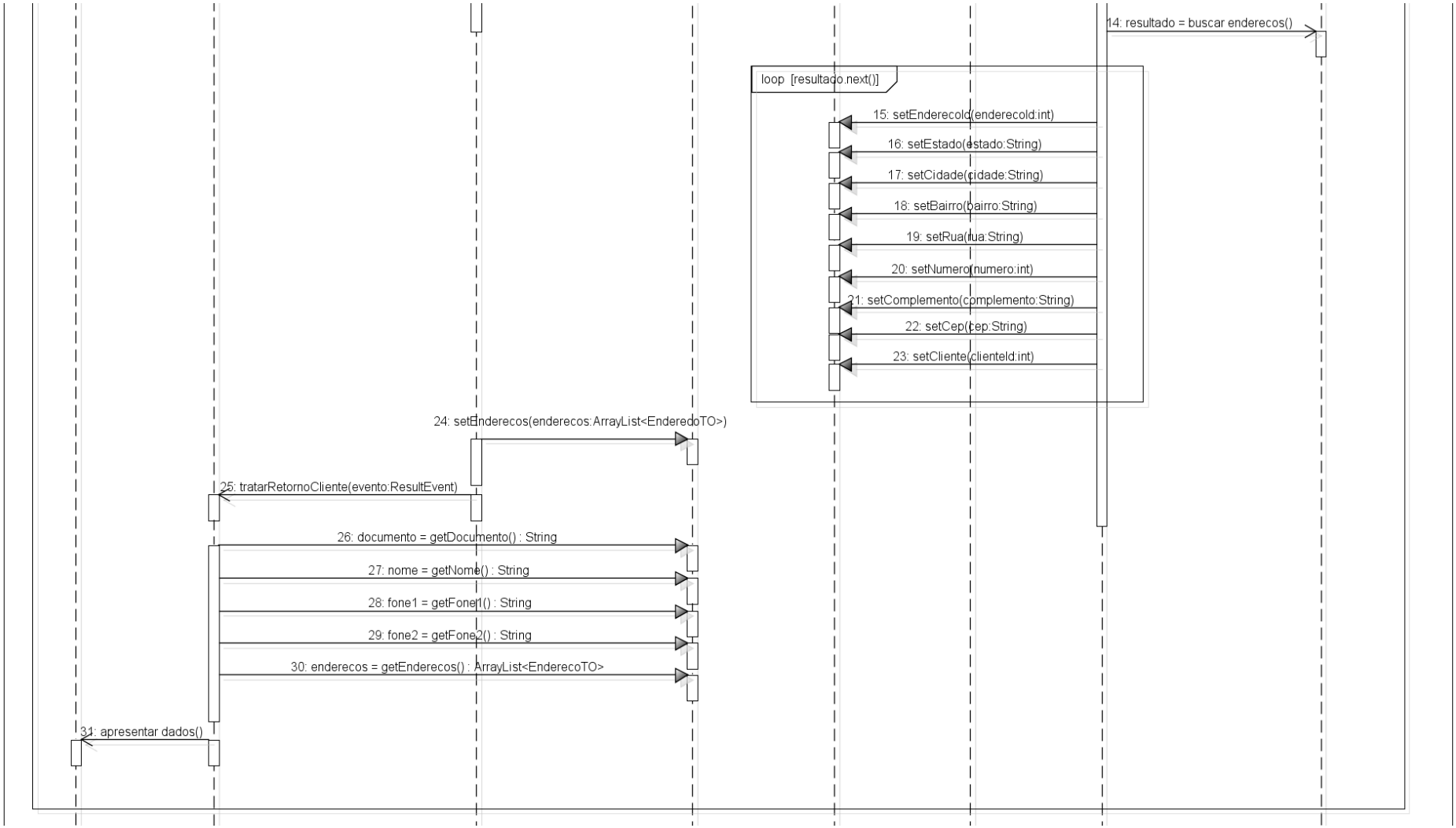


Figura 52 - Parte 2: Diagrama de Seqüência Consultar Cliente

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA ALTERAR CLIENTE

O caso de uso “Alterar Cliente” foi desenvolvido para permitir ao usuário atualizar informações de cadastro, como telefones de contato, e adicionar novos locais à lista de endereços associados a um determinado cliente. Primeiramente, o cliente é localizado; os detalhes da busca estão no tópico anterior, e o subcaso de uso correspondente é referenciado na Figura 53.

Em seguida, pode-se modificar os telefones de contato ou o nome do cliente. O documento não é editável, pois parte-se do pressuposto que é uma informação imutável.

Os endereços já cadastrados não podem ser modificados, para que se preserve um histórico dos mesmos, mas novos endereços podem ser inseridos na base através desta funcionalidade. Como já havia sido explicado no subcaso de uso “Cadastrar Cliente”, o atendente não precisa digitar todos os dados do logradouro; basta que informe o CEP, e o sistema localiza as informações complementares. Na Figura 54, aparece mais uma vez retratada a possibilidade de um CEP retornar mais de um local; neste caso, como já foi esclarecido, o usuário seleciona o bairro e rua desejados em uma janela do tipo *popup*.

Por fim, a Figura 55 conclui o processo, detalhando o fluxo de salvamento das informações na base de dados.

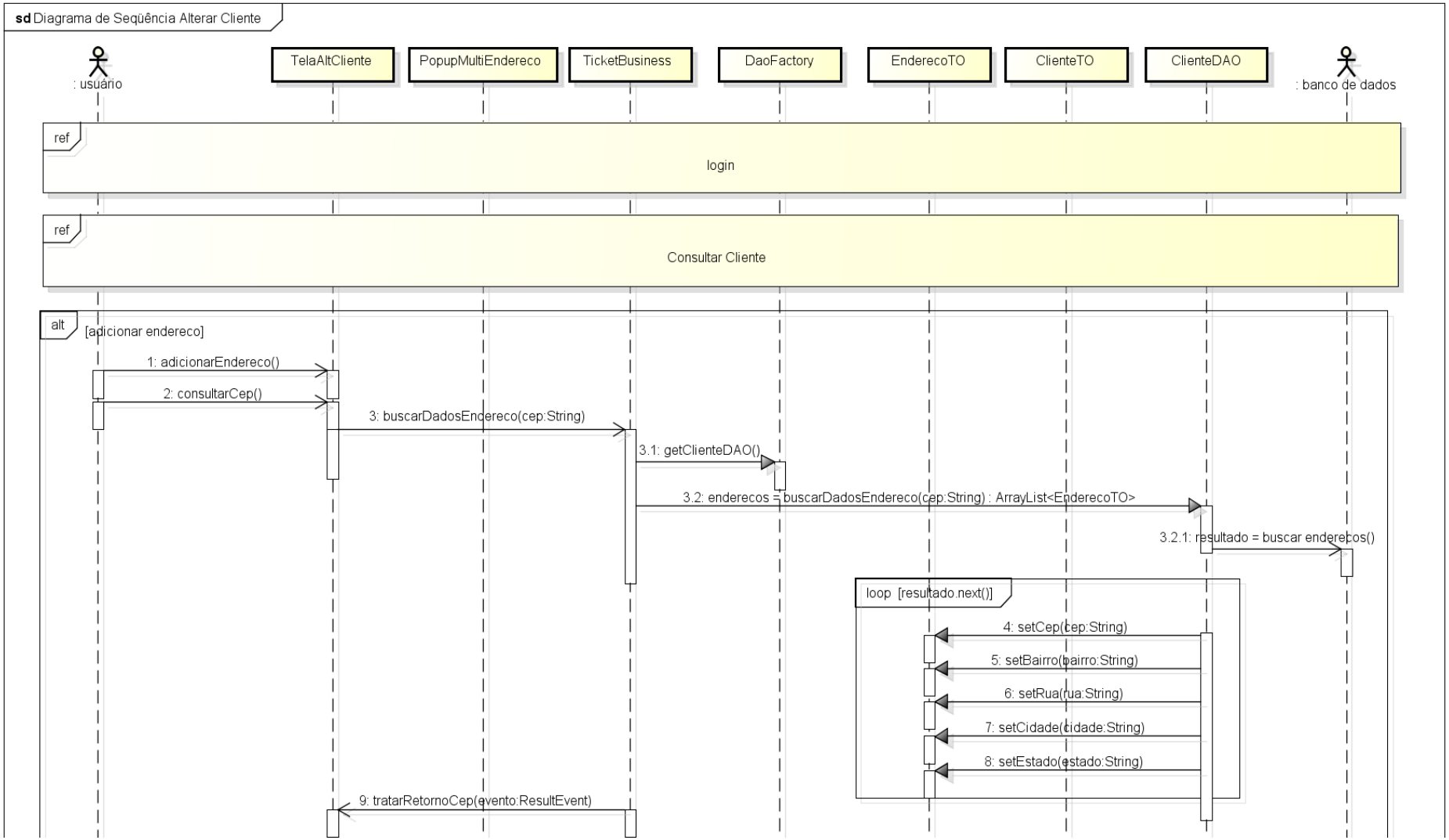


Figura 53 - Parte 1: Diagrama de Seqüência Alterar Cliente

Fonte: Autoria própria.

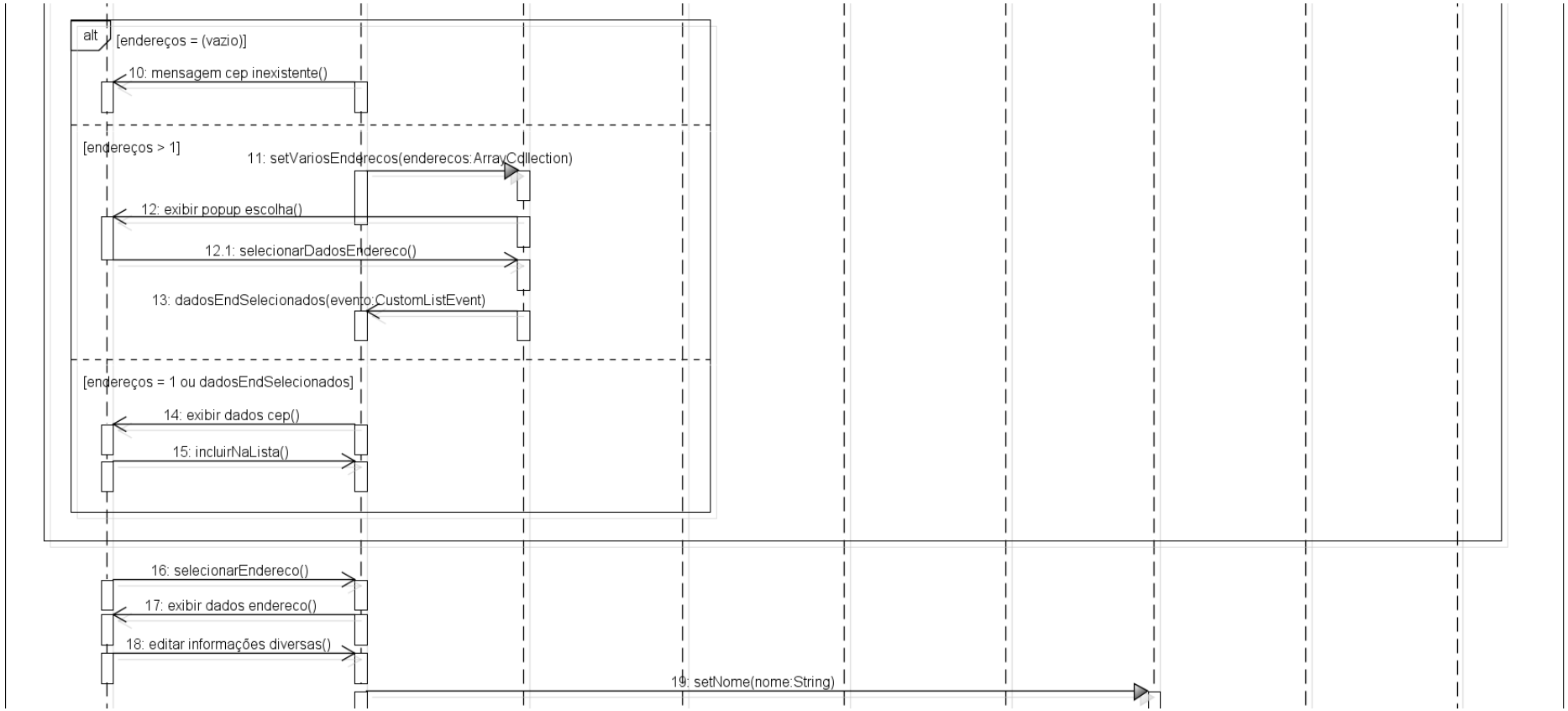


Figura 54 - Parte 2: Diagrama de Seqüência Alterar Cliente

Fonte: Autoria própria.

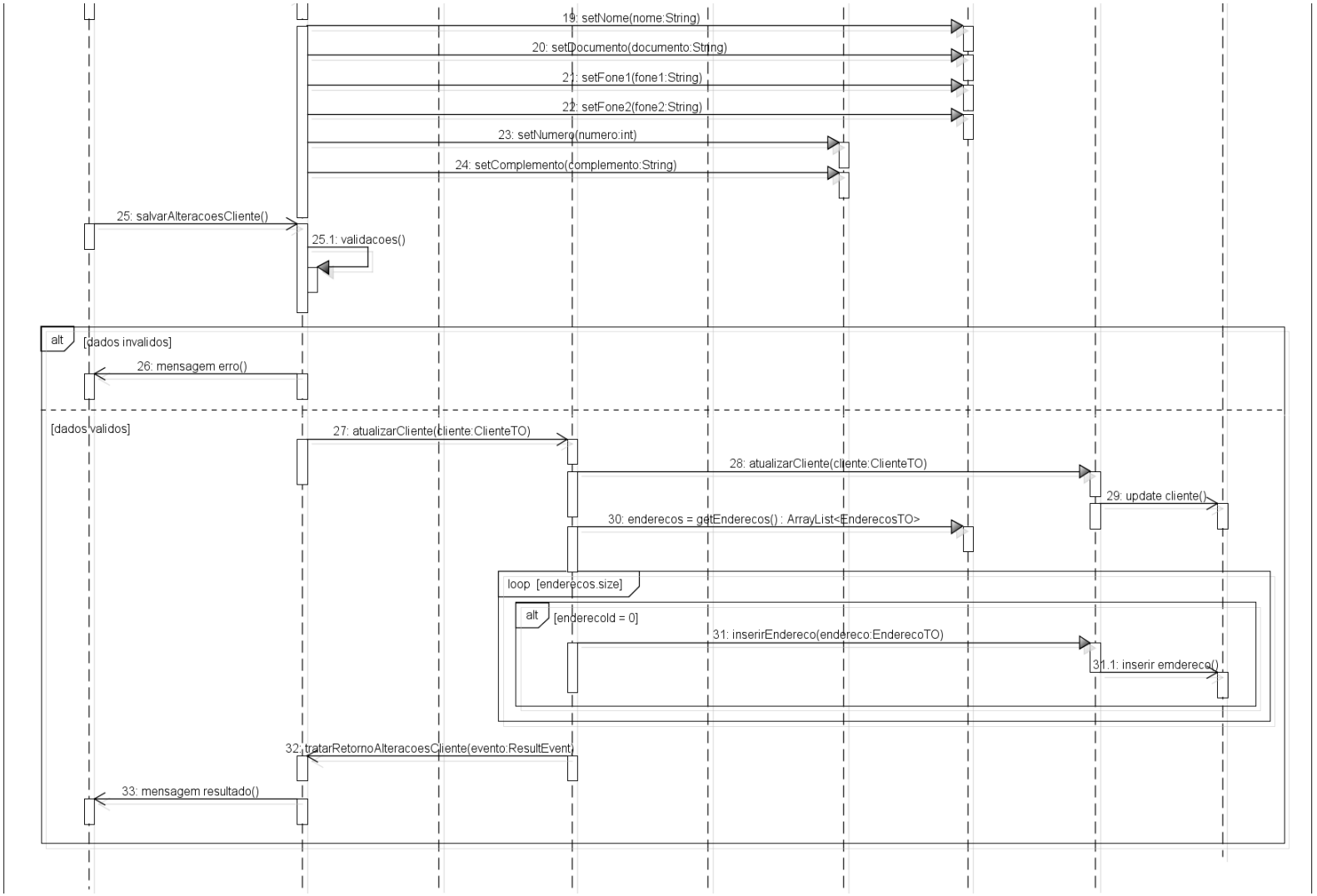


Figura 55 - Parte 3: Diagrama de Seqüência Alterar Cliente

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA CONSULTAR TICKET

O caso de uso “Consultar Ticket”, retratado na Figura 56, na Figura 57 e na Figura 58, além de representar por si só uma funcionalidade do Ulixes, também atua como um subcaso de uso em “Alterar Ticket” e “Despachar Ticket Manualmente”. Logo no início, é oferecida ao usuário a escolha de buscar o ticket pelo número do mesmo (id do registro na tabela Ticket) ou pelo protocolo associado. Dois métodos distintos estão preparados para recuperar do banco de dados os resultados, seja qual for o parâmetro.

Uma vez fornecido o id da solicitação de atendimento ou o número do protocolo, o ticket correspondente (se existir) é exibido para o usuário em detalhes. Os dados trazidos não são apenas os que constam na tabela Ticket, mas também quaisquer informações complementares relacionadas, como o cliente a ser atendido, o endereço vinculado à solicitação, a categorização do ticket (tipo, subtipo e detalhe), status, prioridade e observações do técnico ou do operador que cadastrou a solicitação.

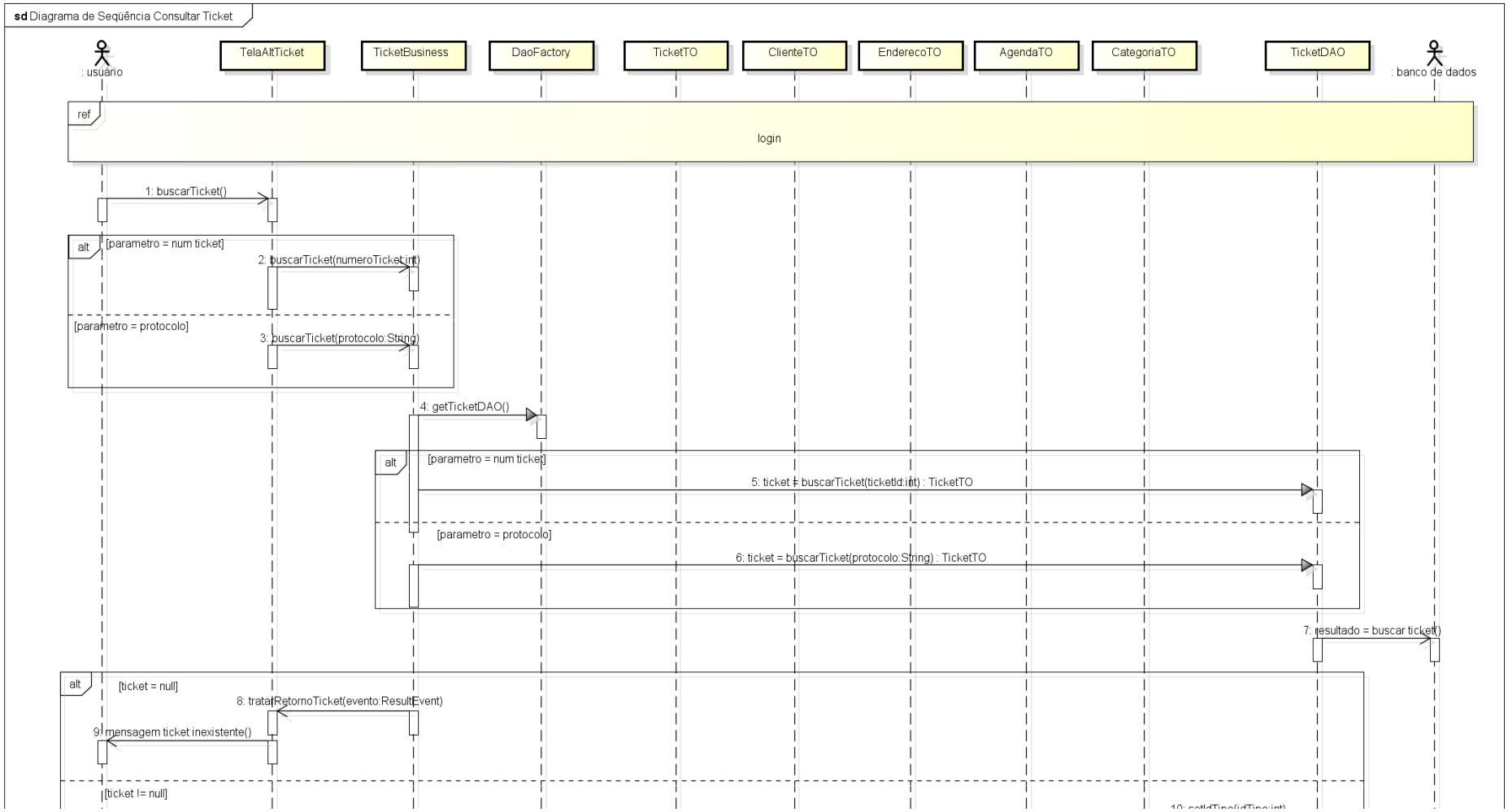


Figura 56 - Parte 1: Diagrama de Seqüência Consultar Ticket

Fonte: Autoria própria.

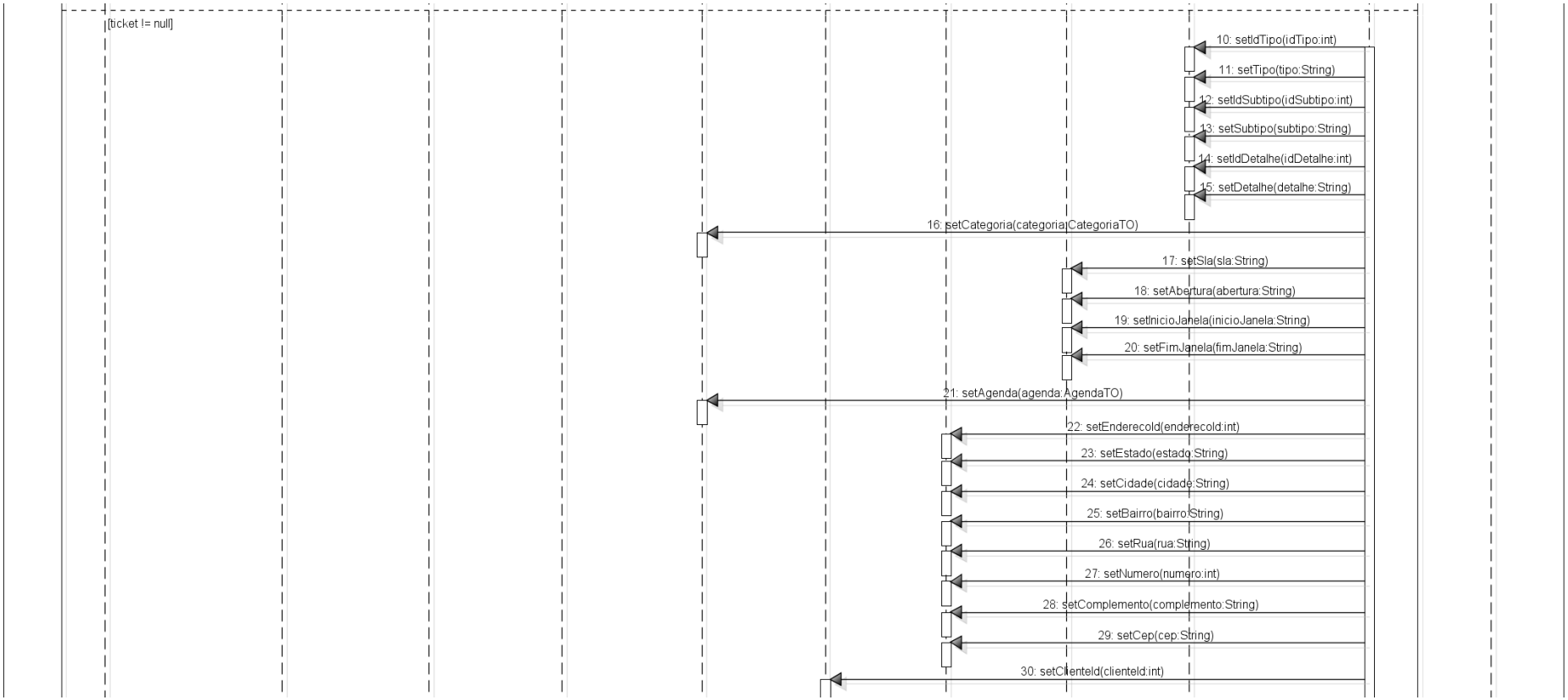


Figura 57 - Parte 2: Diagrama de Seqüência Consultar Ticket

Fonte: Autoria própria.

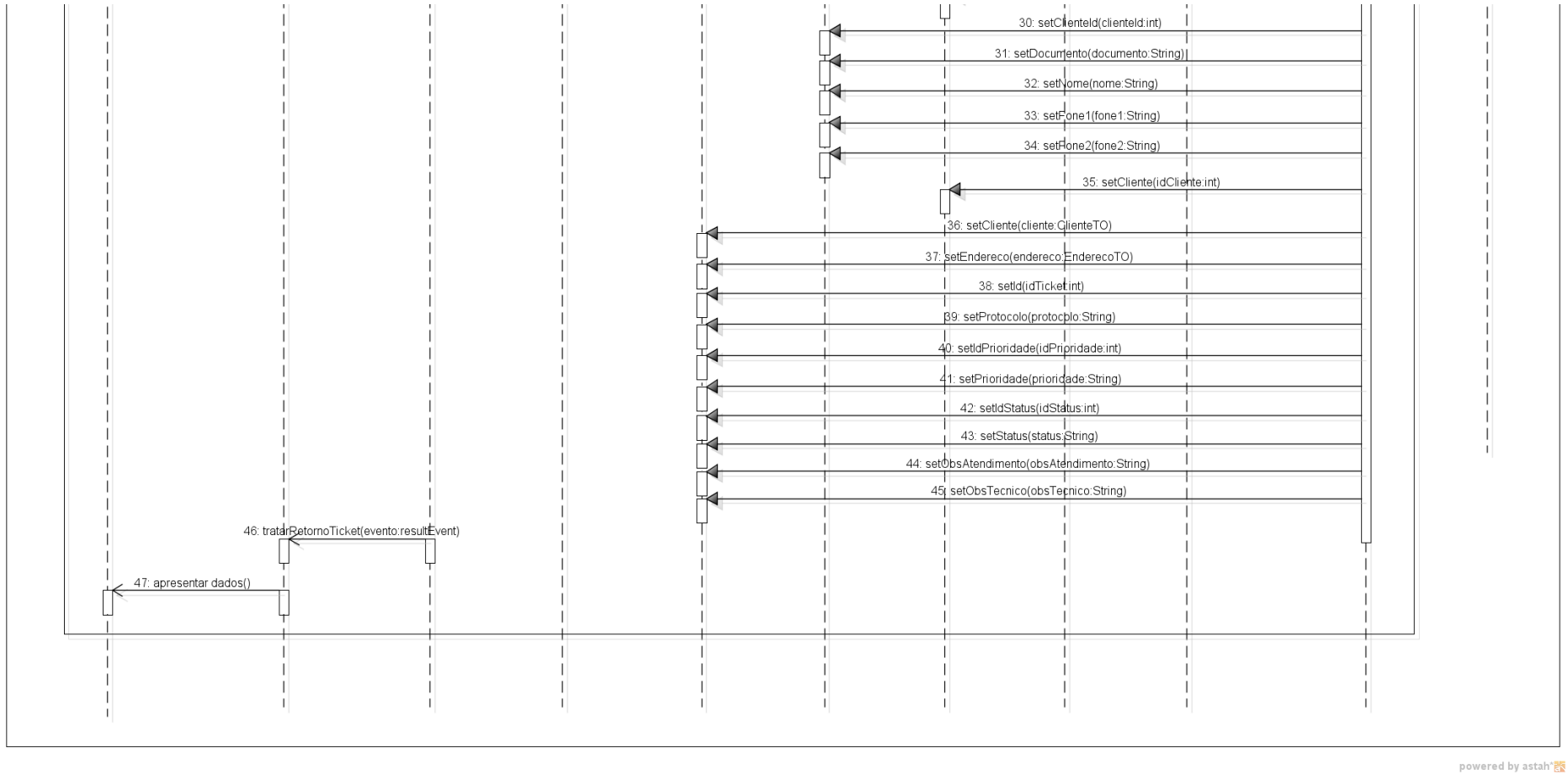


Figura 58 - Parte 3: Diagrama de Seqüência Consultar Ticket

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA INSERIR TICKET VIA SERVIÇO

Este caso de uso é extremamente similar ao Inserir Ticket, uma vez que grande parte do algoritmo se mantém de um para outro. A diferença mais evidente entre ambos, como atesta a Figura 59, é o agente que dispara o processo – não mais um usuário, e sim uma outra aplicação.

Sendo um serviço criado com a finalidade de permitir a integração do Ulixes com outros sistemas, a funcionalidade "Inserir Ticket Via Serviço" não tem etapa de login. O *software* externo que está se comunicando com o Ulixes envia um xml adequadamente formatado, e os dados do mesmo são armazenados em um objeto de valores.

Se fornecido um id de cliente e um id de endereço já existentes, o ticket é associado a ambos. Se os ids não forem preenchidos no xml, mas houverem as informações necessárias para o cadastro de um novo cliente e de um novo endereço, a classe de negócios se encarregará de invocar os métodos apropriados no objeto de acesso a dados.

Uma vez que não é possível restringir de maneira segura os parâmetros enviados via xml – diferente de uma interface de tela, na qual menus *drop-down* e recursos semelhantes limitam os *inputs* do usuário a conjuntos pré-determinados – a classe de negócios deste caso de uso tem uma carga maior de validações. Como explicita a Figura 60, em caso de erro não é apresentada uma mensagem visual da forma como ocorria no caso "Inserir Ticket"; ao invés disso, é lançada uma exceção, com uma mensagem explicitando a natureza da falha ocorrida (ausência de algum parâmetro obrigatório, valor de parâmetro inválido, etc.).

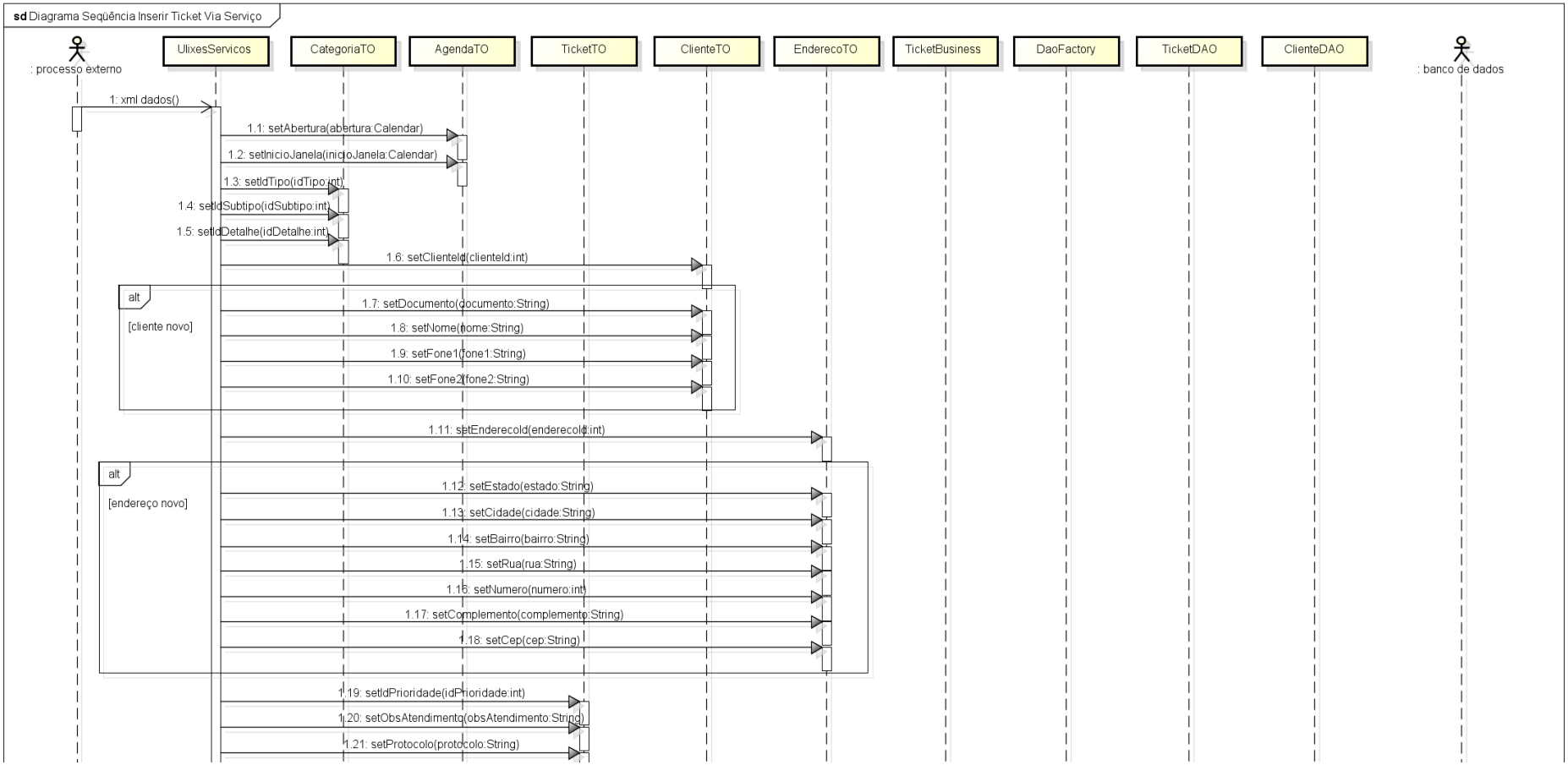


Figura 59 - Parte 1: Diagrama de Seqüência Inserir Ticket Via Serviço

Fonte: Autoria própria.

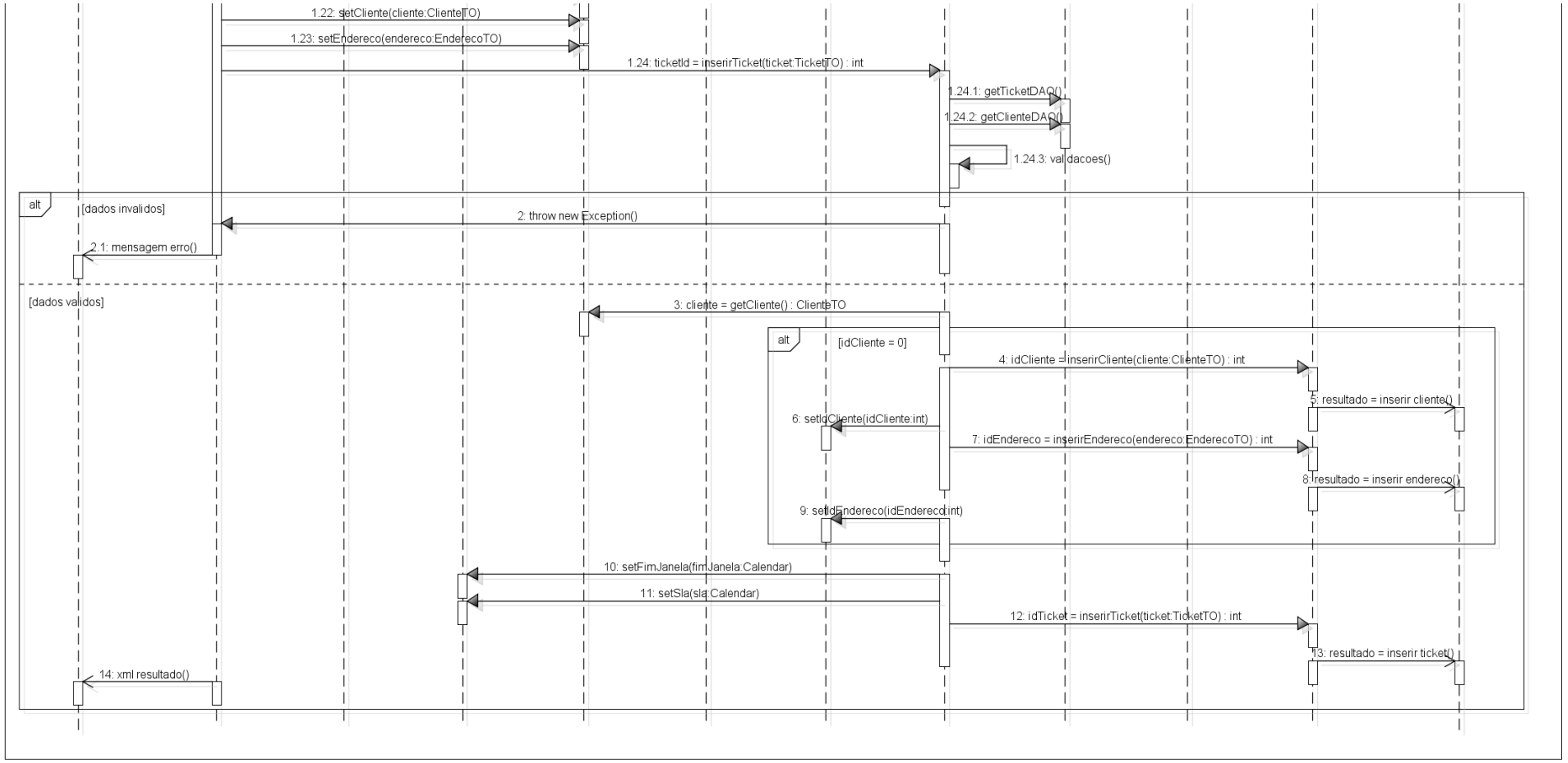


Figura 60 - Parte 2: Diagrama de Seqüência Inserir Ticket Via Serviço

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA ALTERAR TICKET

O caso de uso “Alterar Ticket” foi concebido principalmente para a modificação do status do ticket e do eventual reagendamento da janela de atendimento; é possível observar na Figura 61, no entanto, que ele permite a alteração de alguns outros dados.

Após a consulta da solicitação de serviço através de id ou protocolo associado – funcionalidade já detalhada em um tópico anterior – o usuário pode mudar o protocolo vinculado ao ticket, alterar a categoria (tipo, subtipo e detalhe), acrescentar observações técnicas ou gerais, definir uma nova prioridade ou agendá-lo para uma nova data. Não está explícito no diagrama, mas existem algumas restrições na interface; por exemplo, um ticket que tenha a janela modificada tem o status automaticamente alterado para “Reagendado”. A nova data também não pode ser inferior ao horário atual do sistema mais três horas; isso impede que um ticket seja reagendado para um certo intervalo depois que os tickets a serem atendidos neste mesmo intervalo já foram despachados a campo.

Após as validações essenciais, se o ticket estiver de acordo com as regras utilizadas na verificação, será atualizado no Ulixes. Na Figura 62 está ilustrada essa etapa; conforme mostra o diagrama, o SLA e o fim da janela de atendimento são recalculados conforme as mudanças efetuadas. Se o novo status do ticket for “Encerrado” ou “Cancelado”, a data de fechamento também é preenchida com o próprio horário do sistema.

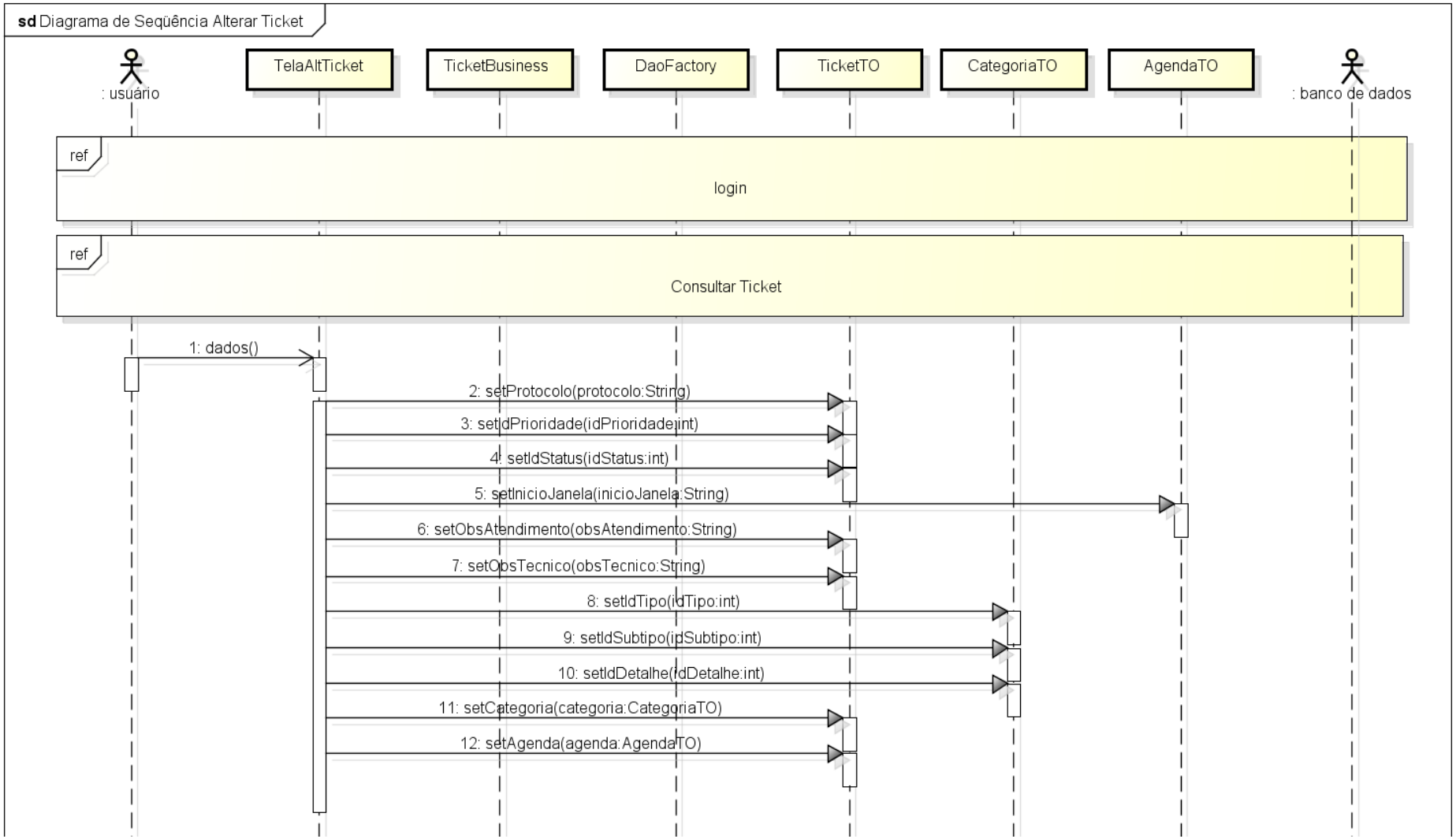


Figura 61 - Parte 1: Diagrama de Seqüência Alterar Ticket

Fonte: Autoria própria.

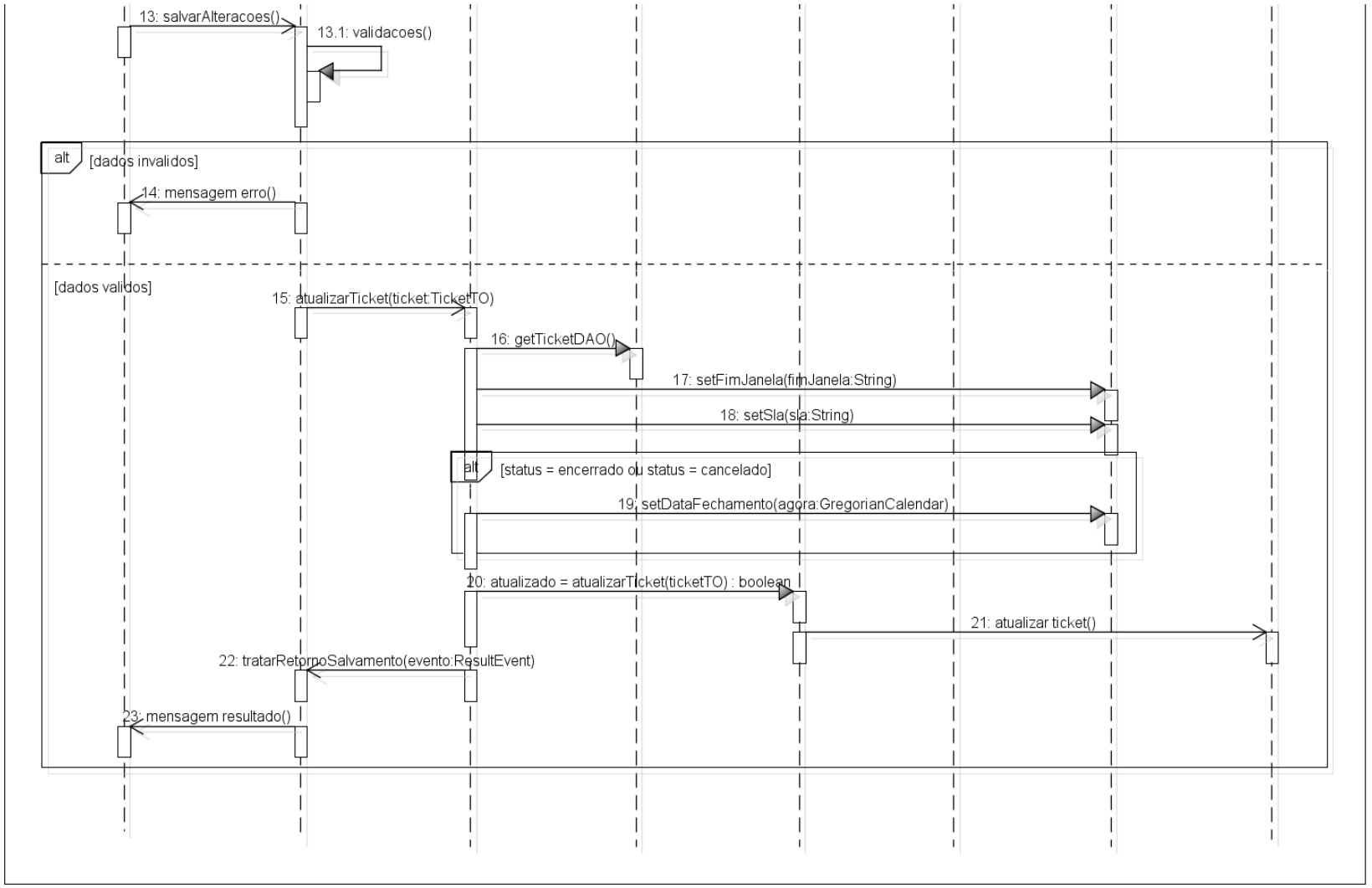


Figura 62 - Parte 2: Diagrama de Seqüência Alterar Ticket

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA ALTERAR TICKET VIA SERVIÇO

Assim como o “Inserir Ticket Via Serviço” está para o caso de uso “Inserir Ticket”, este está para o “Alterar Ticket”. A lógica de ambos é extremamente similar, e as alterações que podem ser feitas no ticket são as mesmas que a funcionalidade do subtópico anterior oferece, conforme se pode observar na Figura 63.

Como já foi frisado, sendo difícil restringir o *input* que o serviço recebe, e sem o recurso de algumas validações disponíveis já na interface, preferiu-se aumentar a quantidade de verificações neste caso de uso. A Figura 64 mostra novamente o lançamento de uma exceção em caso de erro; a exceção traz uma mensagem descritiva do problema ocorrido durante a corroboração das informações (parâmetro inválido ou ausente, etc.).

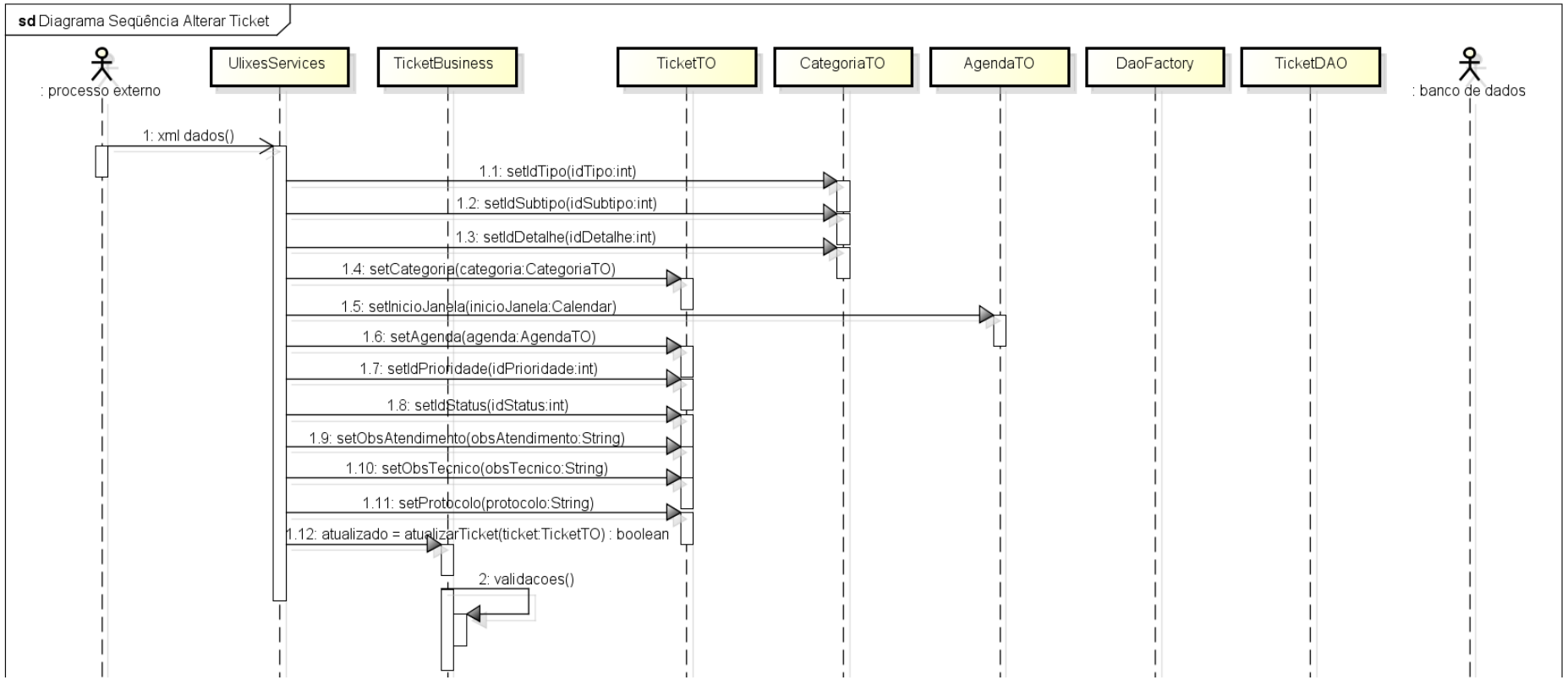


Figura 63 - Parte 1: Diagrama de Seqüência Alterar Ticket Via Serviço

Fonte: Autoria própria.

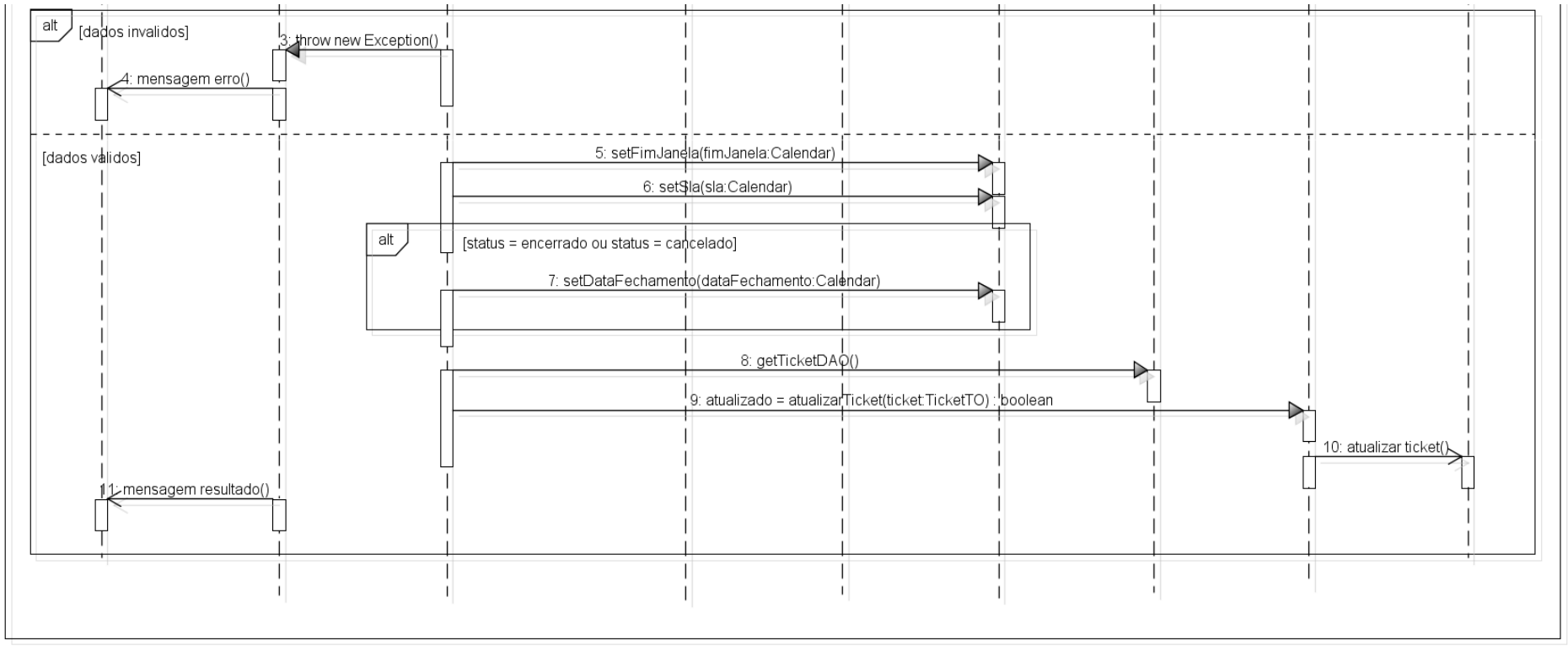


Figura 64 - Parte 2: Diagrama de Seqüência Alterar Ticket Via Serviço

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA CONSULTAR CLIENTE VIA SERVIÇO

Para que os processos externos possam passar os parâmetros corretos no caso de uso “Inserir Ticket Via Serviço”, quando se desejar criar uma solicitação para um cliente já existente, foi criado como auxiliar a funcionalidade “Consultar Cliente Via Serviço”. Mais uma vez, o ator que inicia o fluxo visto no diagrama da Figura 65 é um processo externo (como um *software* de CRM, por exemplo).

O parâmetro de busca a ser enviado no xml de chamada do *webservice* é o documento do cliente – cpf se for pessoa física, e cnpj em caso contrário. Se o dado corresponder a algum cliente cadastrado na base de dados, ele retornará informações básicas do cliente, e uma lista de um ou mais endereços – lembrando que o relacionamento entre as entidades Cliente e Endereço é de 1 para N. A Figura 66 mostra essa possibilidade.

O retorno é um xml com os mesmos atributos do objeto de valores ClienteTO. Se não houver registro com o documento fornecido, não ocorre erro, o serviço apenas retorna o xml vazio.

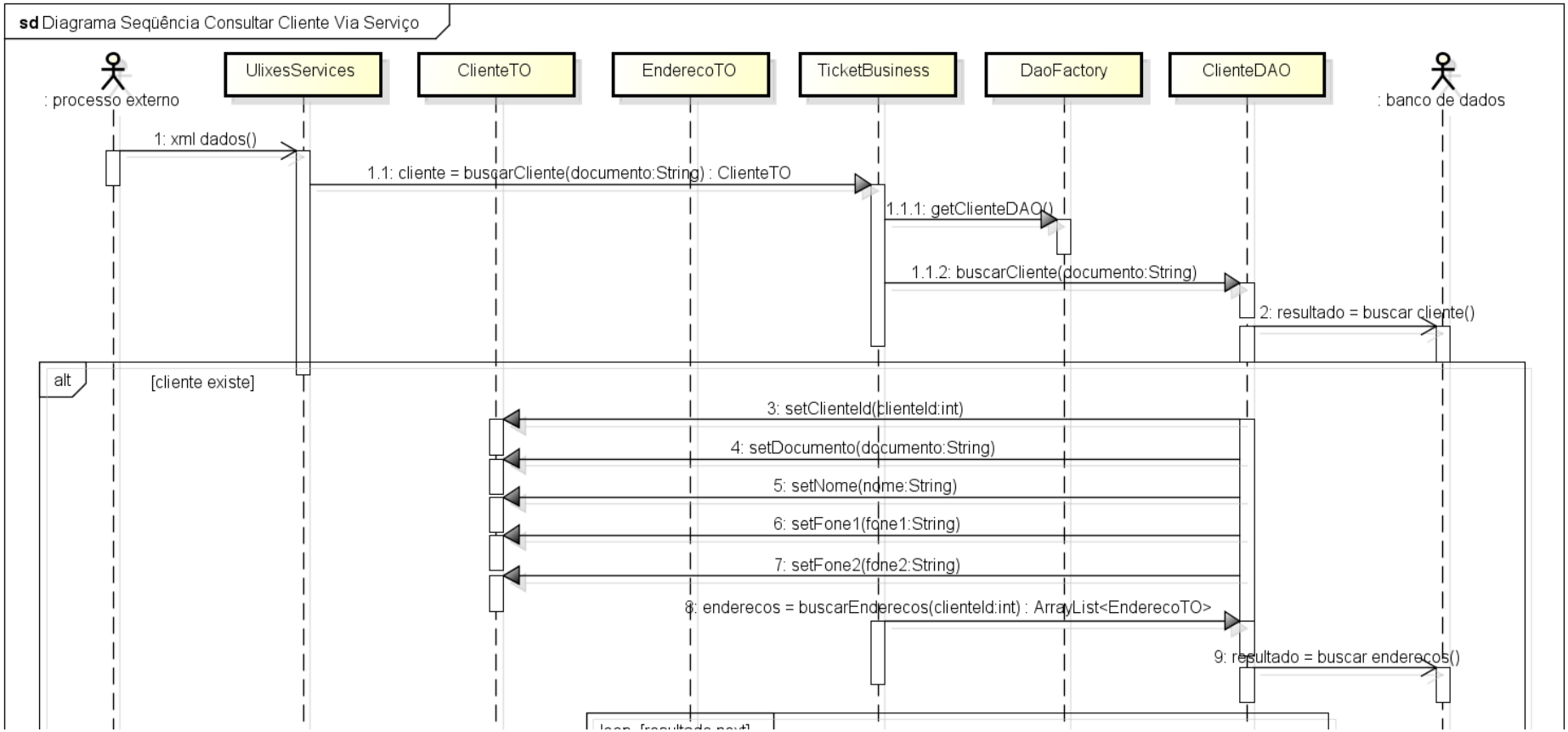


Figura 65 - Parte 1: Diagrama de Seqüência Consultar Cliente Via Serviço

Fonte: Autoria própria.

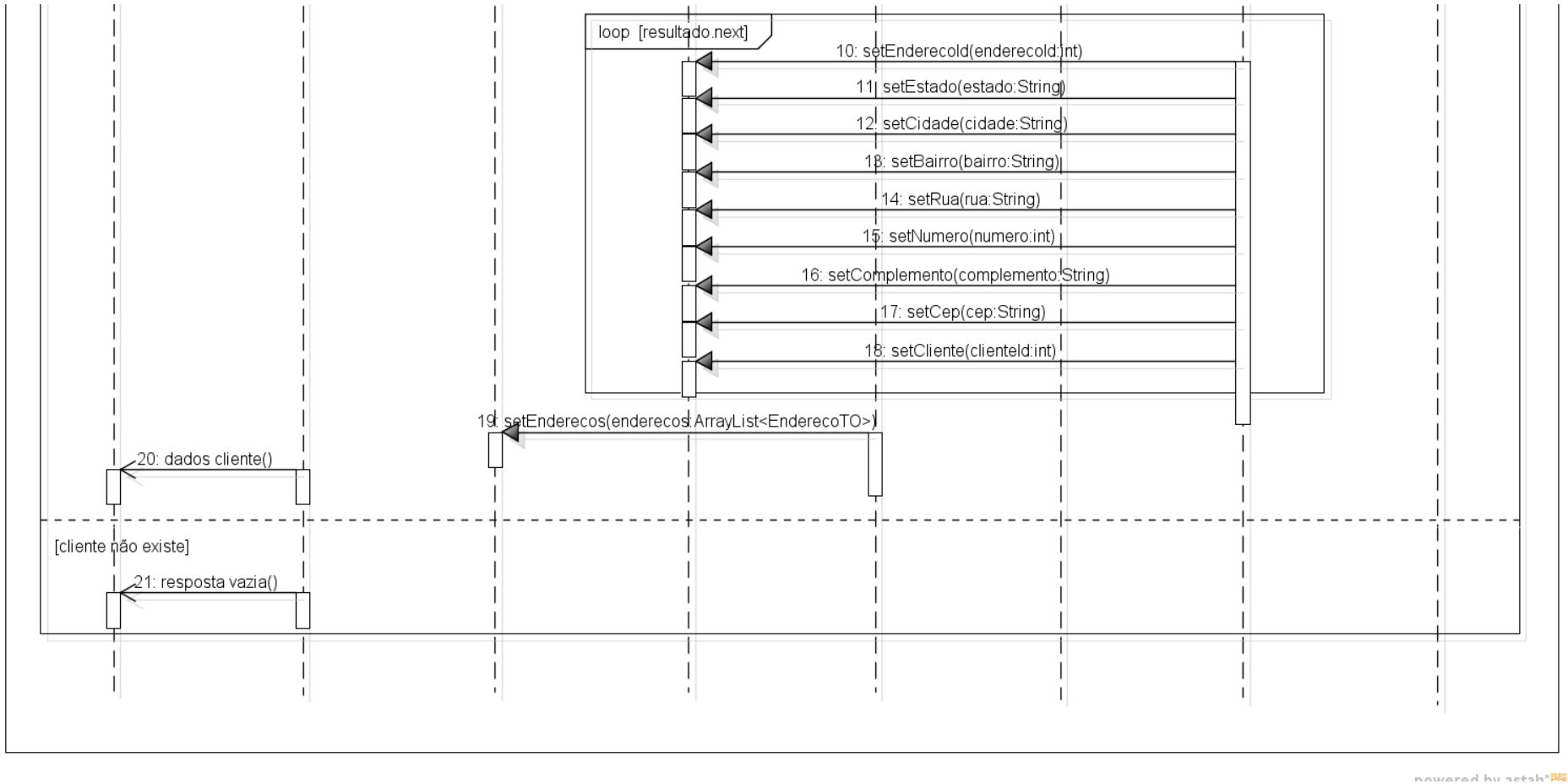


Figura 66 - Parte 2: Diagrama de Seqüência Consultar Cliente Via Serviço

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA DESPACHAR TICKET

Este caso de uso é implementado por um EJB, acionado periodicamente por um *timer*. A notificação do *timer* e a conseqüente chamada do método, vistas na Figura 67, são transparentes para o desenvolvedor.

Assim que é acionada, a classe TicketDispatcher busca todos os tickets com status "Agendado" ou "Reagendado", cujas janelas de atendimento estejam dentro de um período que compreende desde a hora do sistema mais 1h até a hora do sistema mais 1h59min. Assim, o técnico tem tempo hábil de deslocar-se até o lugar do atendimento. O limite de janela dos tickets – nenhum ticket pode ser agendado para um horário menor do que a data atual mais 3h – também foi estabelecido para não conflitar com este processo automático.

Estes tickets são agrupados em um mapa cuja chave é o id da empreiteira. A partir desse momento, o tratamento é feito em um *loop*, sendo que cada iteração representa uma empreiteira específica.

Para a empreiteira que está sendo tratada, são buscados os técnicos disponíveis e seus respectivos limites de turno (hora inicial, hora final), além do número de celular referente ao aparelho de cada um. Estes dados são carregados em uma lista encadeada.

Os tickets vinculados à empreiteira são verificados um a um. Todas as informações necessárias para o atendimento técnico são recuperadas do banco de dados, como evidencia a Figura 68. A lista encadeada de técnicos é percorrida, e procura-se um técnico cujo horário de atendimento seja compatível com a janela agendada para o ticket. Assim que um técnico adequado é localizado, o id deste é associado ao id do ticket através da entidade Atendimento. O status da solicitação é alterada para "Enviado", e a mesma é adicionada à fila de tickets que o técnico tem para solucionar. Finalmente, o técnico é enviado para o fim da lista encadeada – essa medida visa impedir que todos os tickets sejam atribuídos a um único técnico, enquanto os demais permanecem ociosos. Sendo um detalhe relevante do processo, esta medida está ressaltada em um *box* de observação na Figura 69.

Quando todos os tickets já foram vinculados a um profissional técnico, ocorre uma última iteração, desta vez pela lista de técnicos. Um método interno da TicketDispatcher salva no servidor um arquivo html com as informações de cada ticket que foi atribuído a um determinado técnico; em seguida, a url deste arquivo recém-criado é despachada para o celular do técnico em uma mensagem SMS especialmente formatada. Assim, todos os técnicos que estão em seu turno de serviço recebem um SMS com o link através do qual

podem recuperar as solicitações de serviço às quais precisam dar vazão. Uma pequena aplicação de celular serve de interface para que estes usuários interajam com os tickets.

Concluída a lista de técnicos, um novo ciclo inicia para tratar os tickets da empreiteira seguinte, até que o mapa termine.

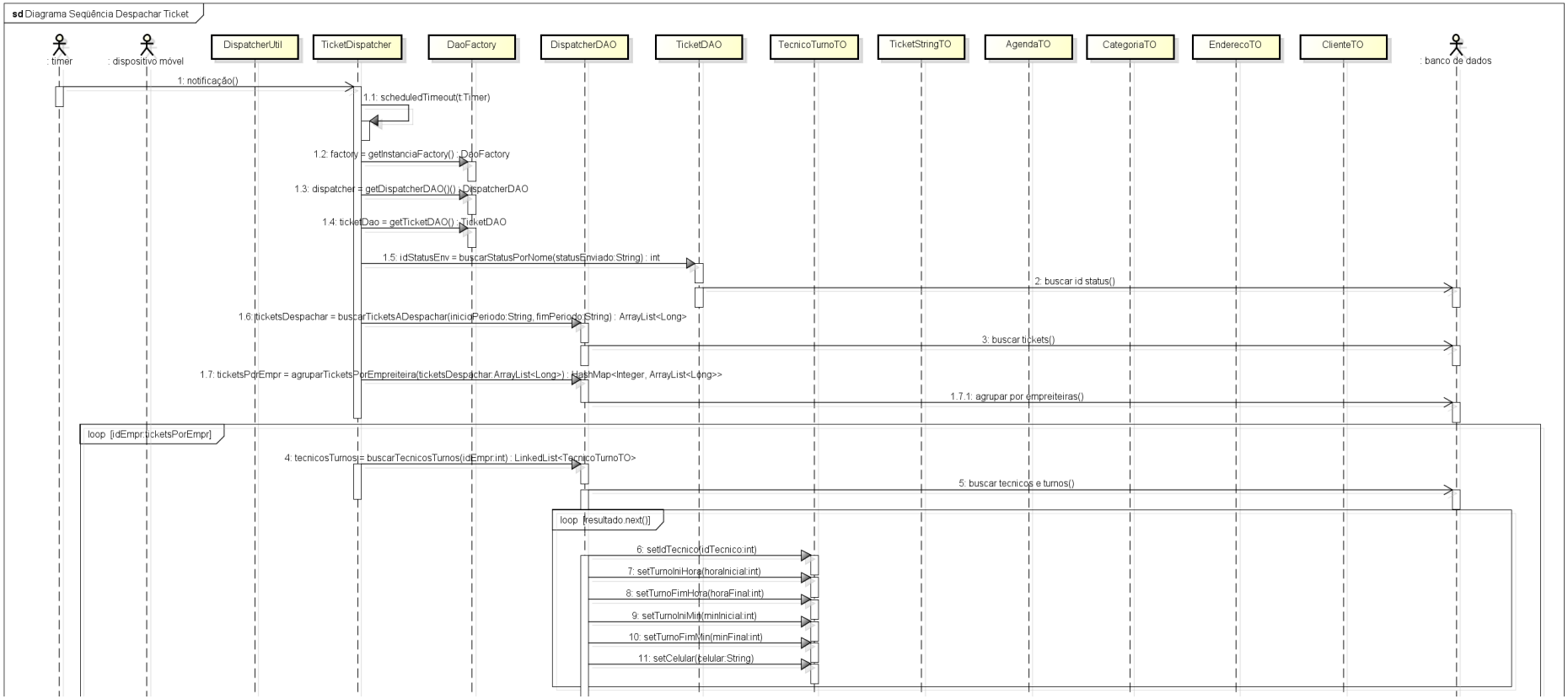


Figura 67 - Parte 1: Diagrama de Seqüência Despachar Ticket

Fonte: Autoria própria.

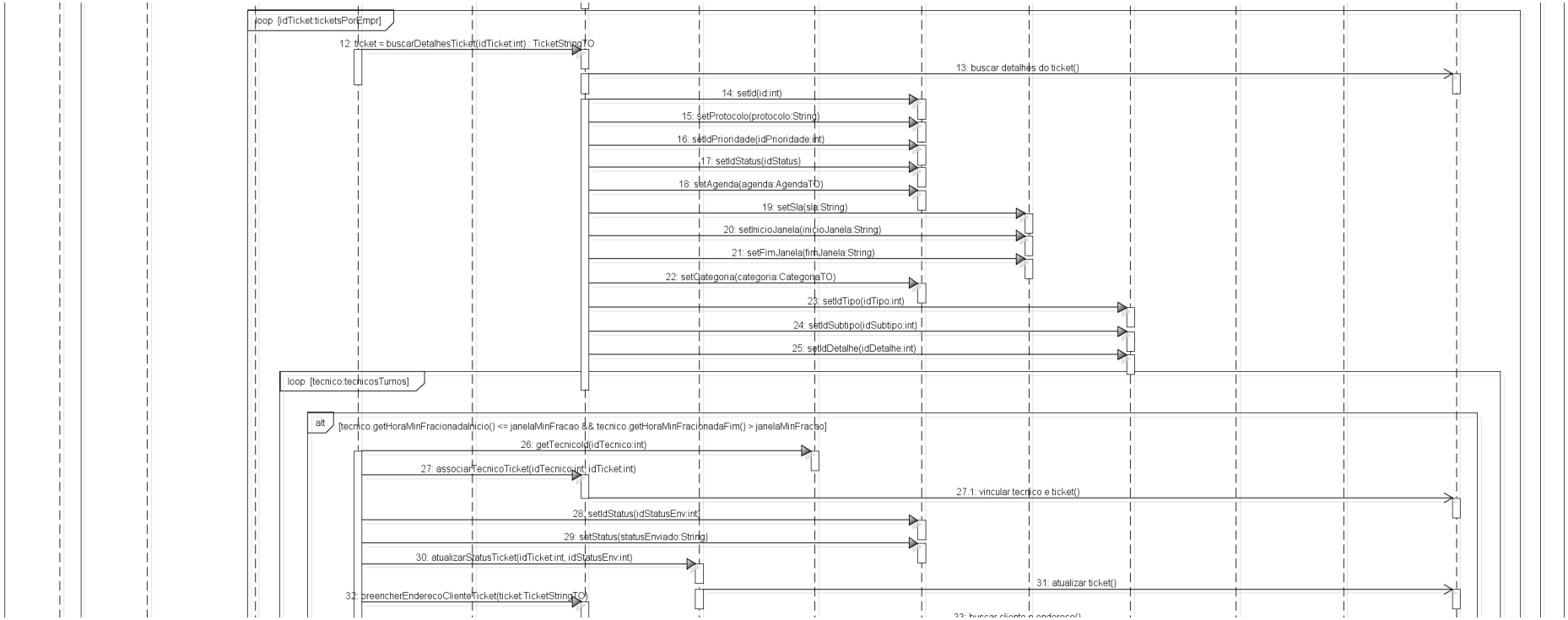


Figura 68 - Parte 2: Diagrama de Seqüência Despachar Ticket

Fonte: Autoria própria.

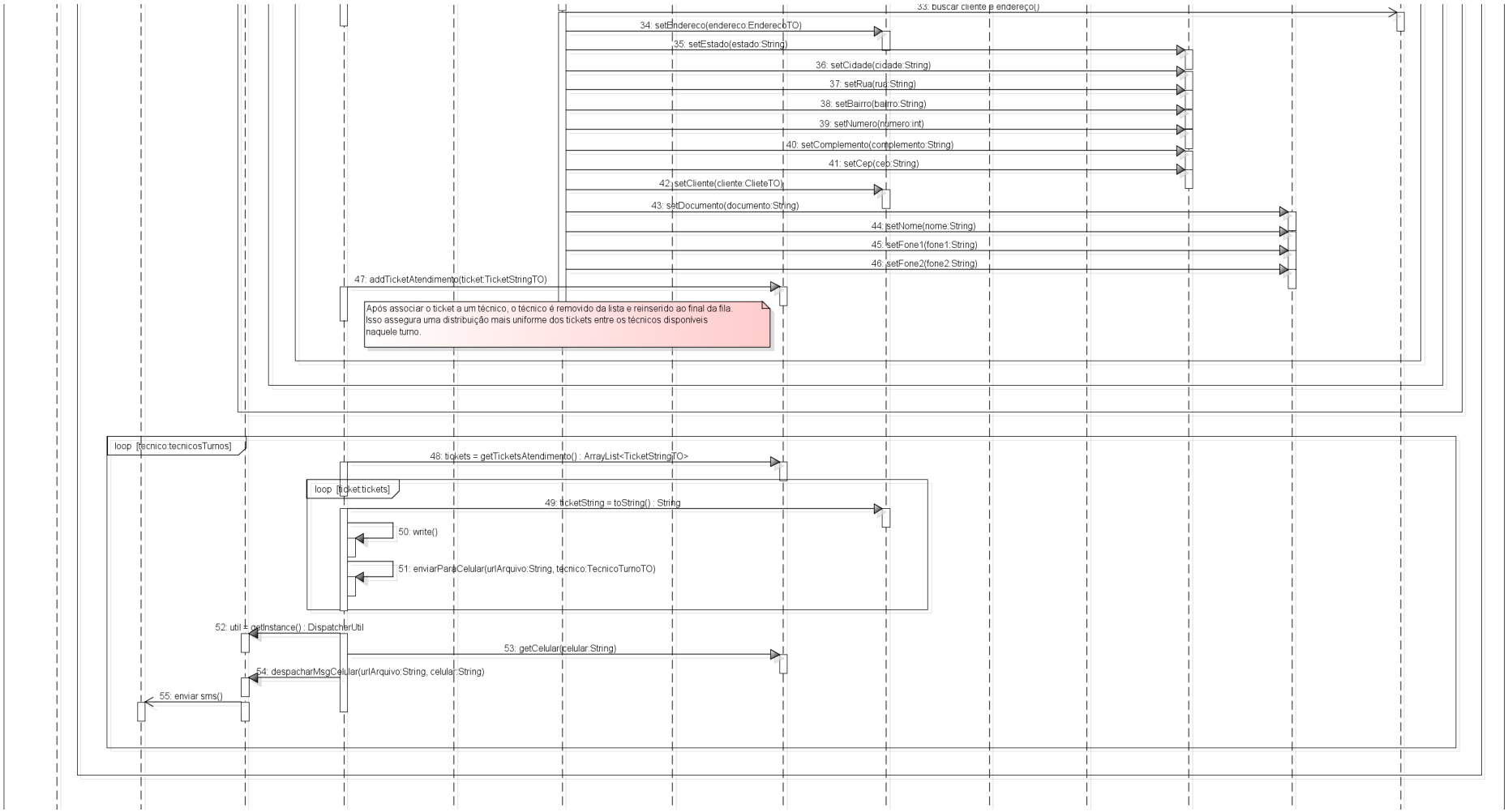


Figura 69 - Parte 3: Diagrama de Seqüência Despachar Ticket

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA DESPACHAR TICKET MANUALMENTE

Essa funcionalidade foi pensada para suporte e para emergências. Em um dado momento, pode ser interessante ou necessário despachar uma solicitação de serviço para um certo técnico, ao invés de esperar que um processo automático o faça. Como foi planejado para ser uma contingência, o caso de uso “Despachar Ticket Manualmente” possui poucas validações e travas de *software*, e deve ser utilizado com parcimônia.

Após o login e o carregamento dos dados de empreiteira que serão utilizados na ferramenta, o usuário consulta um ticket conforme já foi abordado em casos de uso anteriores. Se o ticket consultado não está com status “Agendado” ou “Reagendado”, ele não é apto para ser despachado, e uma mensagem de erro é exibida. Esta primeira etapa está representada na Figura 70.

Sendo o ticket válido, o usuário é solicitado a selecionar uma empreiteira para atendê-lo. A lista de técnicos afiliados à empreiteira é carregada, e obrigatoriamente um deles deve ser escolhido. Por fim, o usuário confirma a operação de despachar o ticket, e a interface executa algumas validações antes de enviar os dados para a camada de negócios.

Logo que o ticket chega à EmpreiteiraBusiness, junto com as informações do técnico responsável, um *request* http é feito para uma *servlet* com os ids de ambos como parâmetro (vide Figura 71). Se a requisição obtiver sucesso, o usuário recebe uma mensagem de confirmação do despachamento.

O motivo de se usar uma *servlet* é a estruturação do Ulixes. Ele está dividido em dois projetos, um com as interfaces de usuário e outro com os *webservices* e *servlets*. Como a lógica de despachamento do ticket e as bibliotecas necessárias para as operações de baixo nível que enviam a mensagem SMS estão no segundo projeto, optou-se por interligar ambos proporcionando assim maior reaproveitamento de código. A chamada para a servlet é justamente esta ligação entre os projetos.

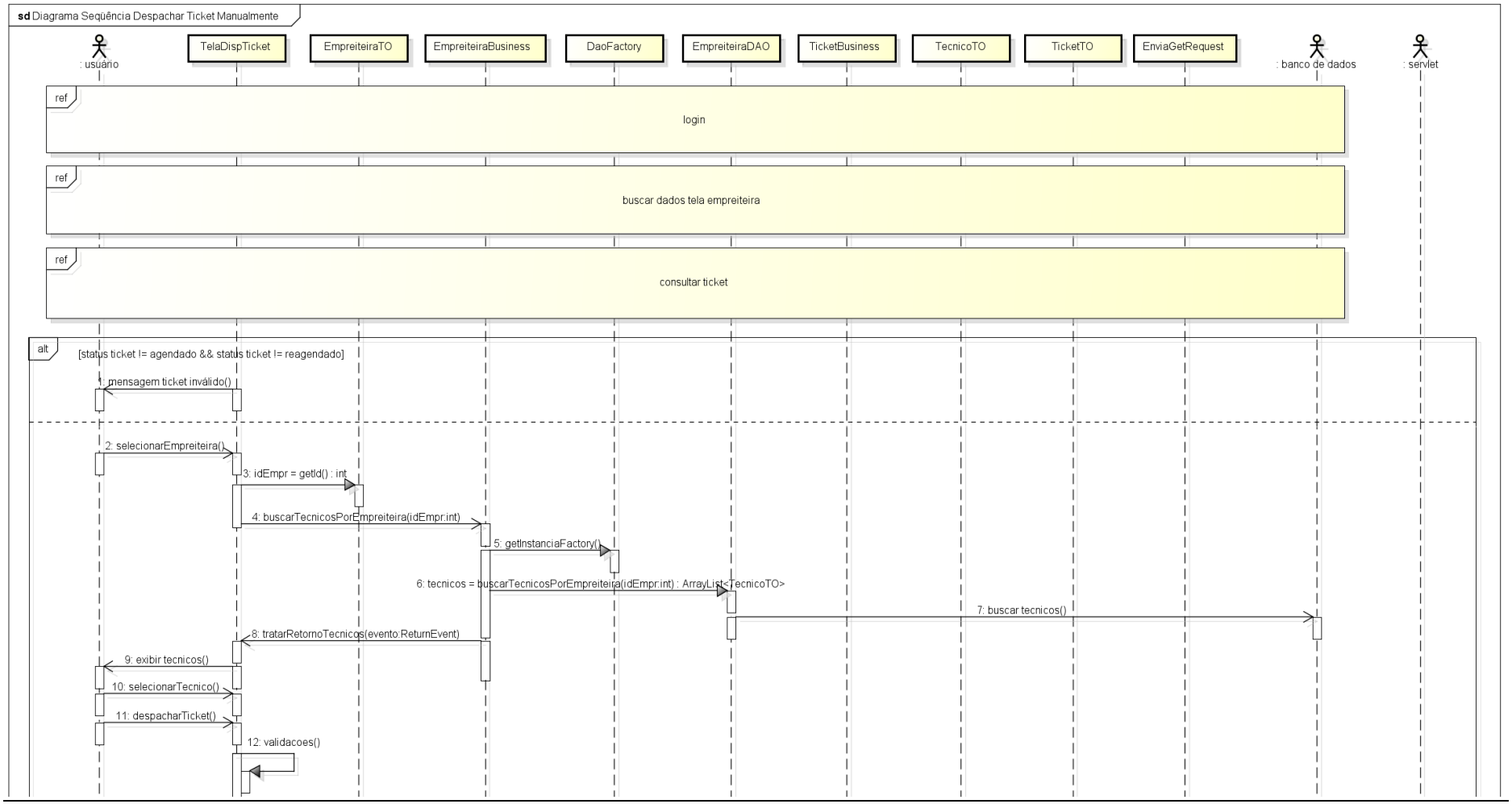


Figura 70 - Parte 1: Diagrama de Seqüência Despachar Ticket Manualmente

Fonte: Autoria própria.

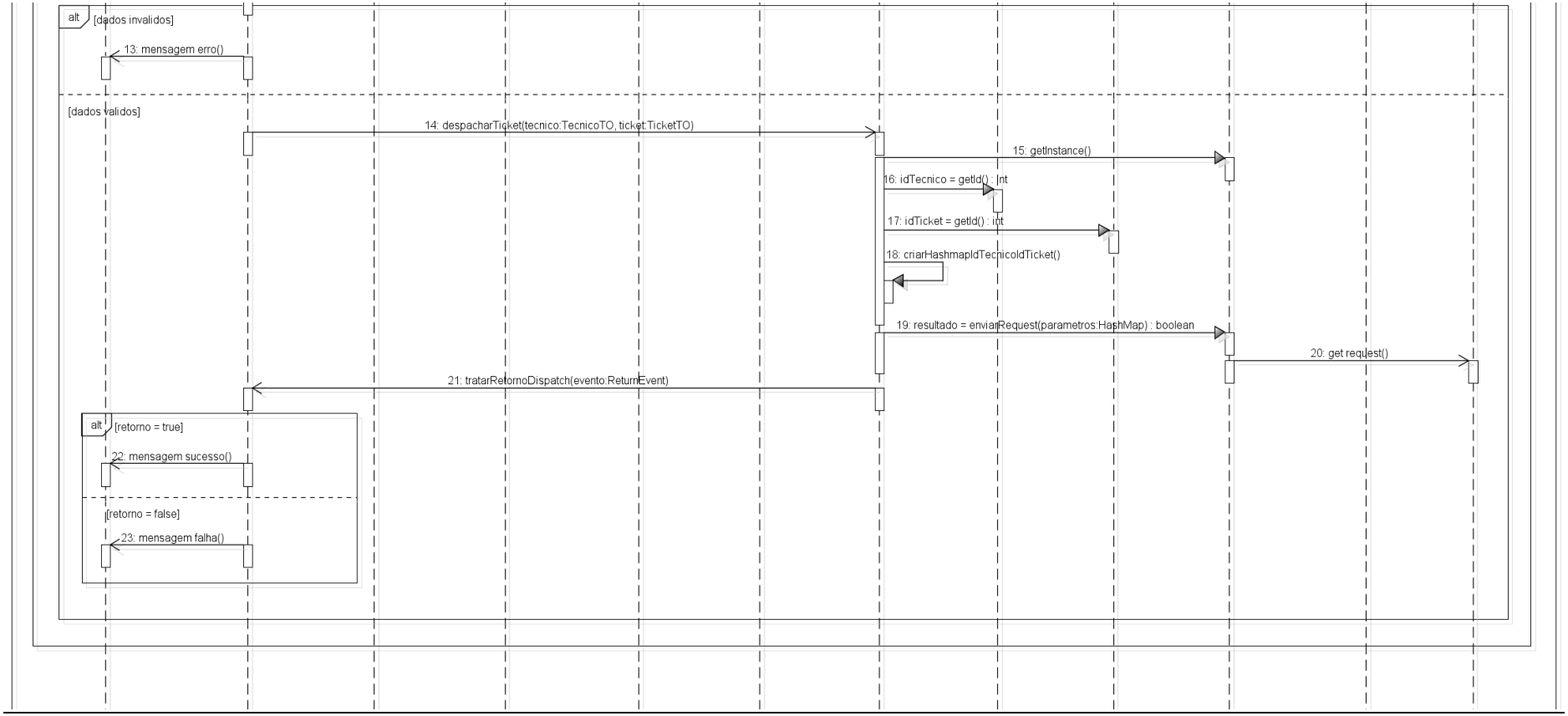


Figura 71 - Parte 2: Diagrama de Seqüência Despachar Ticket Manualmente

Fonte: Autoria própria.

Para melhor entendimento do que ocorre no projeto em que o despachamento do ticket é efetivamente efetuado, foi incluído neste documento um diagrama de seqüência complementar mostrando o percurso do ticket dentro da *servlet*. Ele está dividido na Figura 72 e na Figura 73, e é similar ao caso de uso Despachar Ticket visto em outro tópico.

As maiores diferenças em relação ao processo automático de envio de tickets é que o técnico foi pré-selecionado (portanto não é preciso escolher dinamicamente um técnico) e não há iterações, já que através da tela de despachar tickets só é possível enviar um ticket por vez. A lógica restante é praticamente idêntica, e envolve a geração de uma página html com os dados do ticket, a disponibilização da mesma no servidor, e a emissão de uma mensagem SMS com a url desta página para o aparelho móvel do técnico.

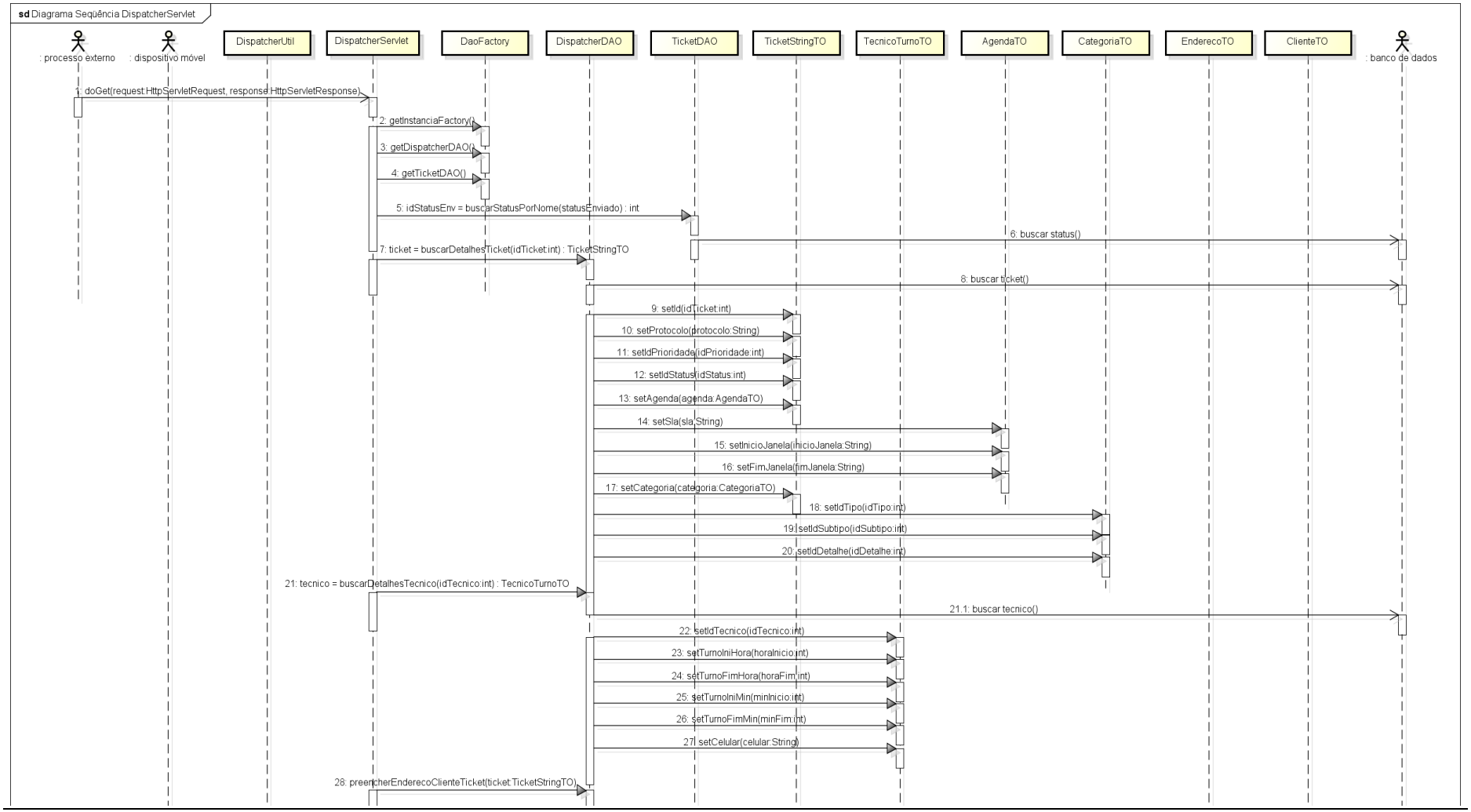


Figura 72 - Parte 1: Diagrama de Seqüência Despachar Ticket Manualmente – *Servlet*

Fonte: Autoria própria.

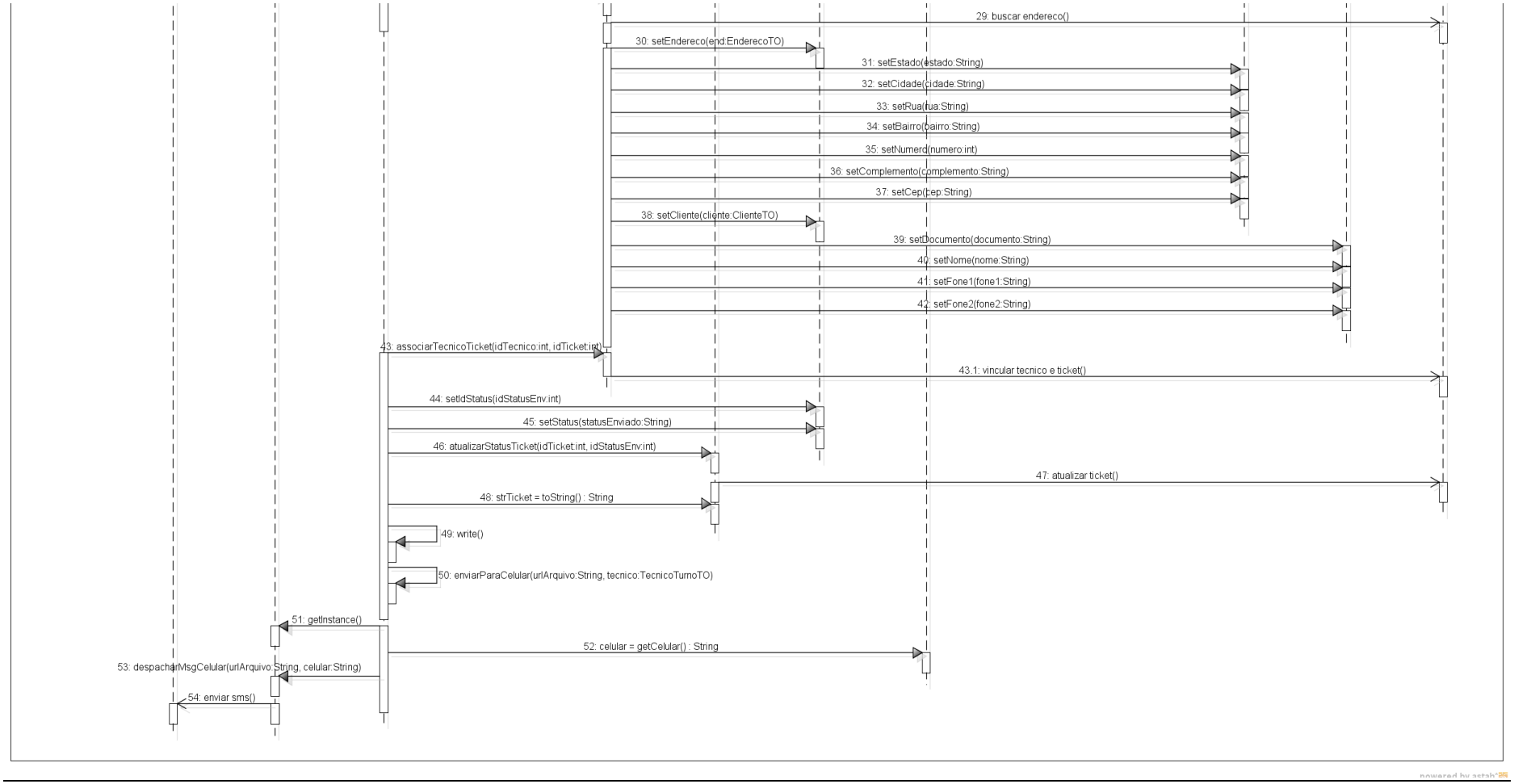


Figura 73 - Parte 2: Diagrama de Seqüência Despachar Ticket Manualmente – *Servlet*

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA EXTRAIR RELATÓRIO

O caso de uso “Extrair Relatório”, apesar de possuir um diagrama de seqüência extenso, tem um algoritmo relativamente simples. Na mesma tela de interface, o usuário tem a opção de escolher oito possíveis filtros para o relatório, além de selecionar o período dentro do qual deseja fazer a pesquisa. A Figura 74 e a Figura 75 mostram as diversas opções, sendo que para cada uma é feita uma busca diferenciada na base de dados.

Na Figura 76, está representado o *loop* de armazenamento dos resultados em uma lista de objetos de valores. Se o usuário escolhe ver um relatório de tickets por tipo, para citar um exemplo, os diversos tipos de ticket e as quantidades referentes a cada um – assim como a porcentagem que representam do total – preencherão esta lista de retorno. Para outros filtros, a lógica é semelhante.

Uma vez que os valores são apresentados para o usuário na forma de “gráficos de pizza”, este pode clicar em uma parcela do gráfico para enxergar um detalhamento dos tickets ali representados. Esta operação é conhecida como *drill-down*, e no Ulixes ela também desencadeia buscas distintas no banco para cada um dos filtros disponíveis. Estes acessos à base aparecem não apenas na Figura 76 mas também na Figura 77. As informações recolhidas são exibidas no formato de tabela, sendo cada linha referente a um ticket que compõe o grupo selecionado.

Para encerrar, a interface possui métodos internos para fazer a exportação da tabela em um arquivo excel, se o usuário assim o desejar.

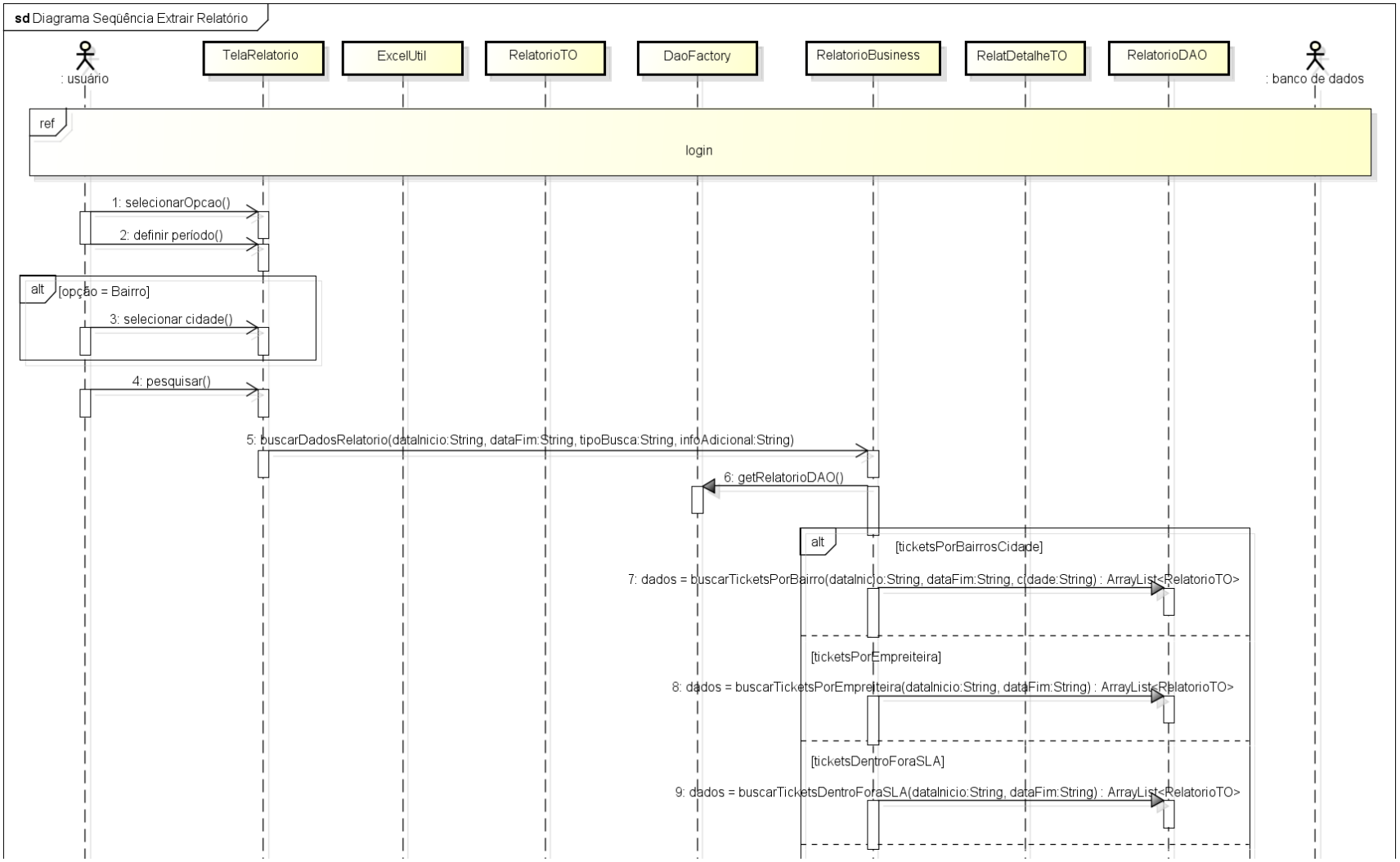


Figura 74 - Parte 1: Diagrama de Seqüência Extrair Relatório

Fonte: Autoria própria.

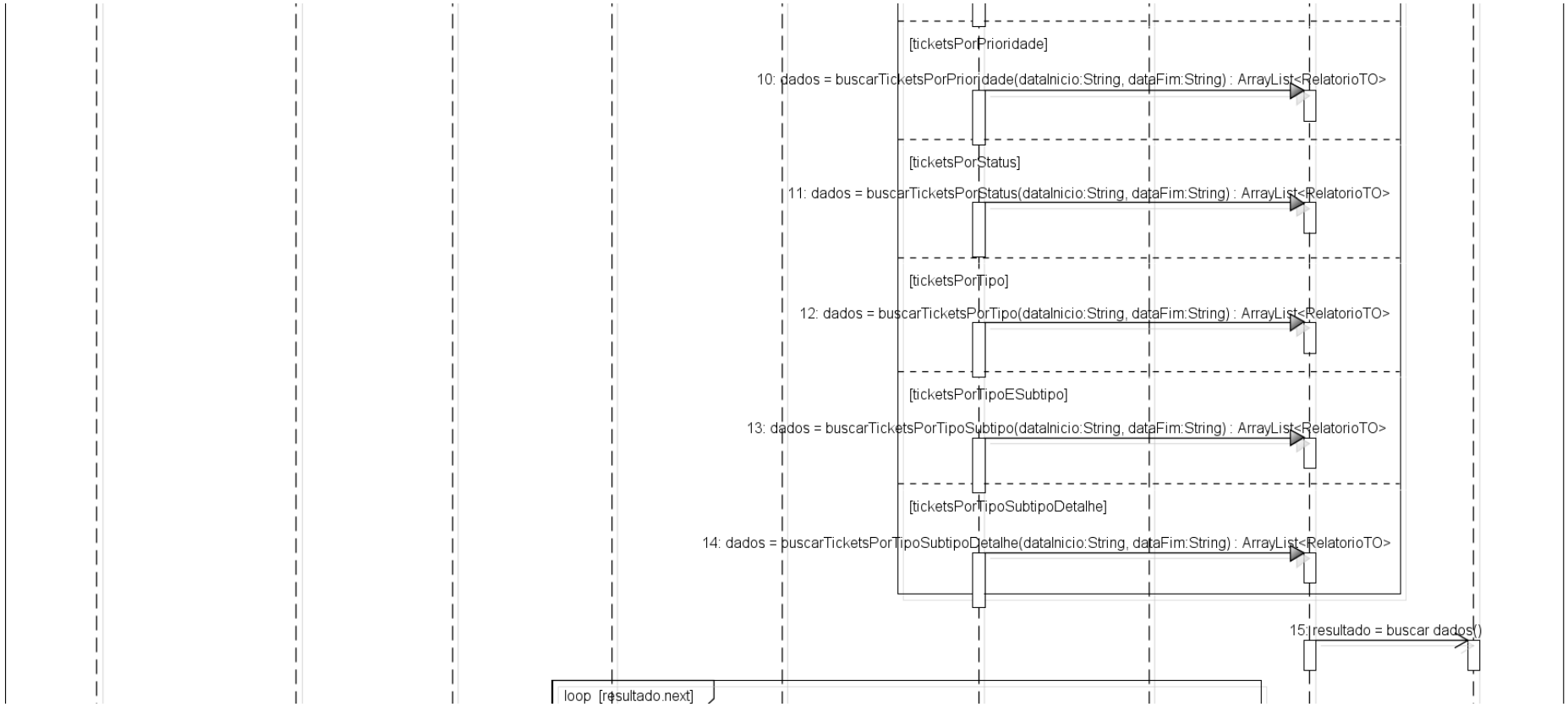


Figura 75 - Parte 2: Diagrama de Seqüência Extrair Relatório

Fonte: Autoria própria.

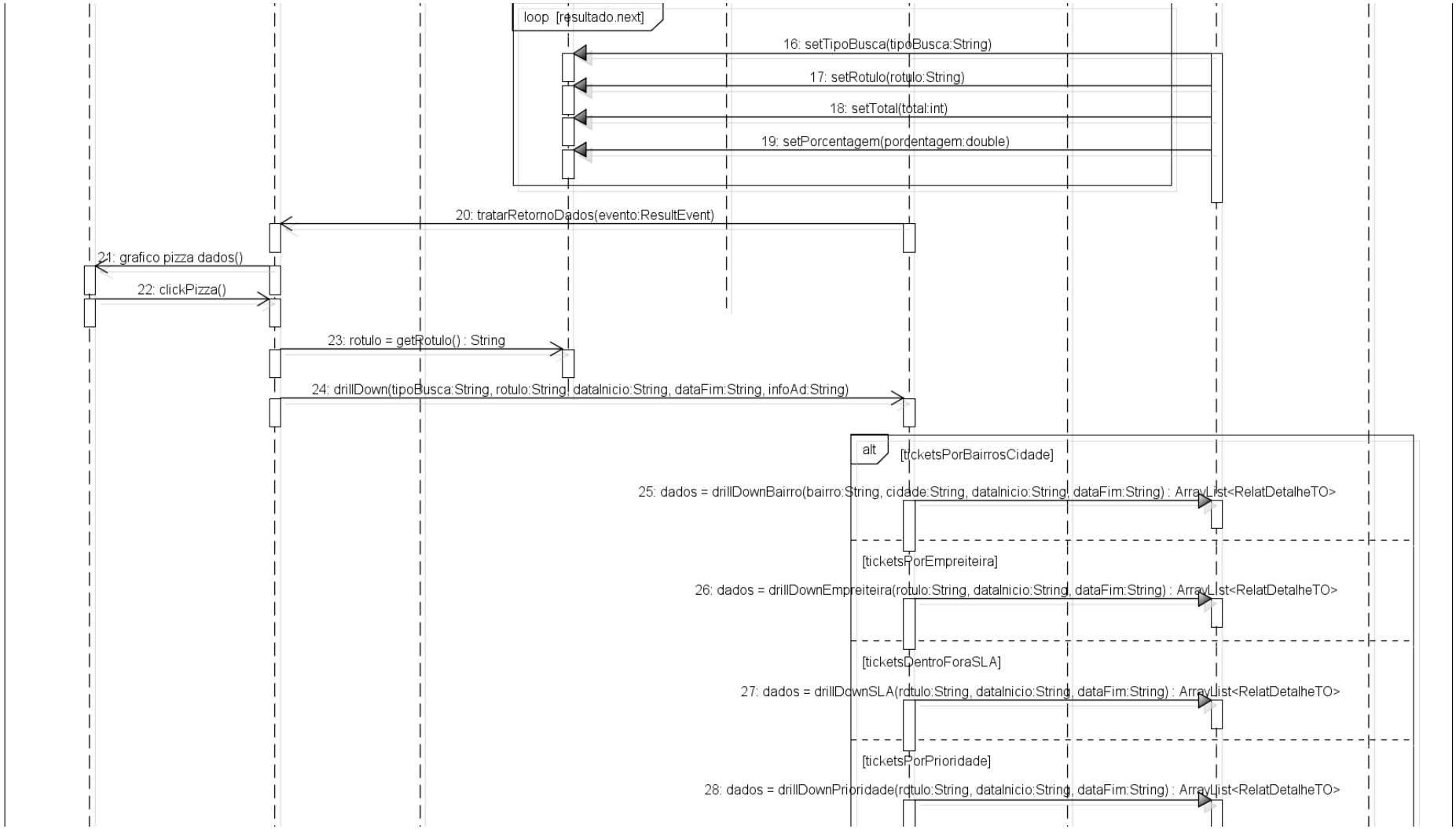


Figura 76 - Parte 3: Diagrama de Seqüência Extrair Relatório

Fonte: Autoria própria.

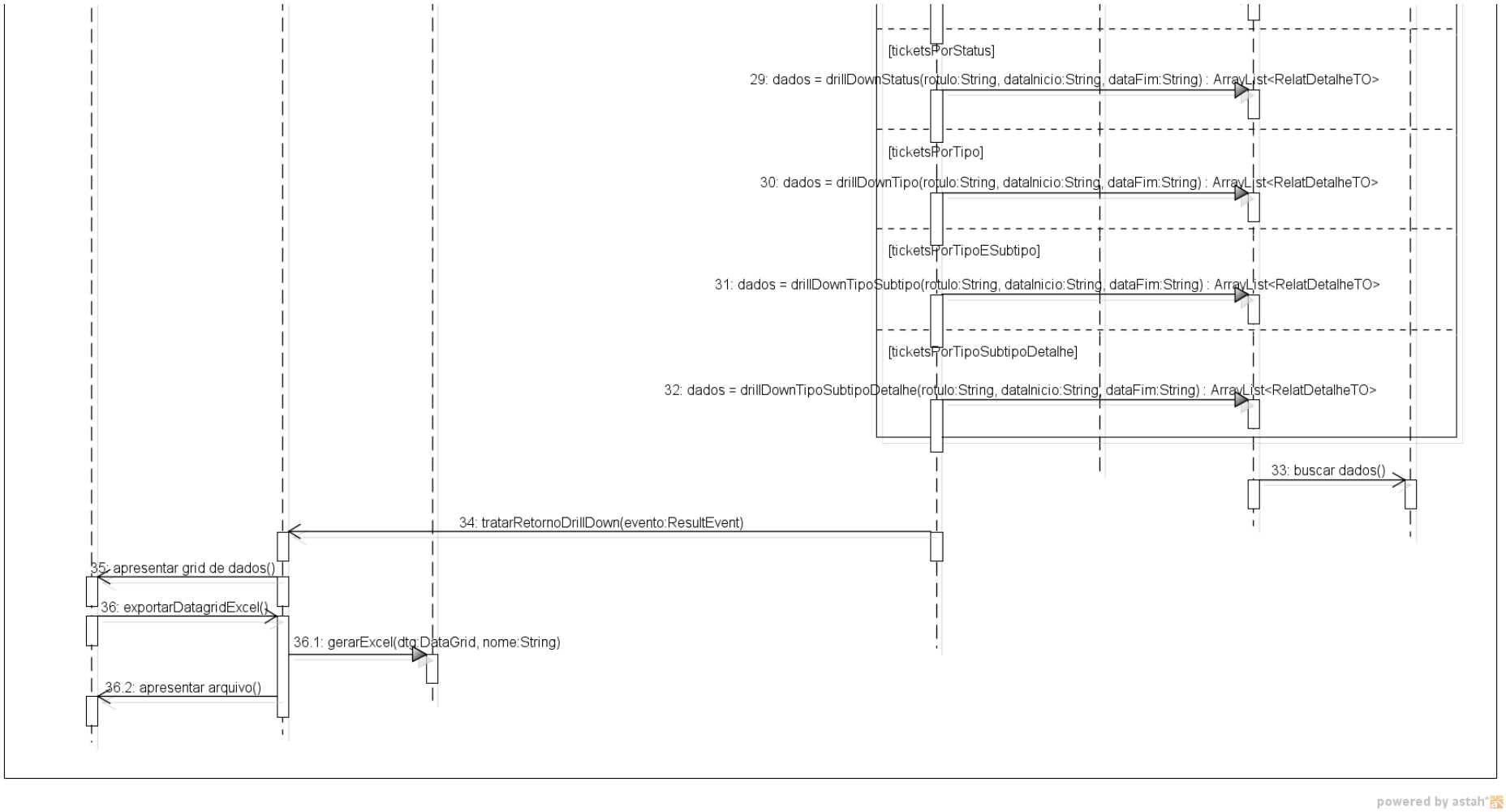


Figura 77 - Parte 4: Diagrama de Seqüência Extrair Relatório

Fonte: Autoria própria.

DIAGRAMA DE SEQÜÊNCIA ALTERAR TICKET VIA DISPOSITIVO MÓVEL

O caso de uso “Alterar Ticket Via Dispositivo Móvel” é um dos mais simples do sistema, ele basicamente descreve o processo de atualização do *status* de um determinado *ticket* pelo técnico em campo.

Na Figura 78 observa-se que após o usuário abrir o aplicativo *ulixesApp* em seu celular e clicar na opção “Tickets”, a tela que possibilita a edição do *status* do *ticket* é apresentada na interface.

Nessa tela, o usuário tem a opção de navegar por todos os chamados que estão pendentes em sua fila de atendimento. Ao encontrar o *ticket* que se deseja atualizar, basta ao técnico um clique simples sobre linha “Status” que um novo diálogo com as opções "Encerrado", "Suspenso Ausência Cliente", "Suspenso Falta de Facilidade" e "Suspenso Pedido de Reagendamento" é apresentado a este. Ao selecionar um novo estado, automaticamente o *ticket* é enviado é despachado para o servidor remoto da aplicação, e se tudo acontecer como o esperado, este é removido da interface e da base de dados do celular do técnico. Caso algo inesperado ocorra, o usuário pode despachar este *ticket* manualmente pela funcionalidade “Enviar Tickets Manualmente” disponível no menu principal do aplicativo.

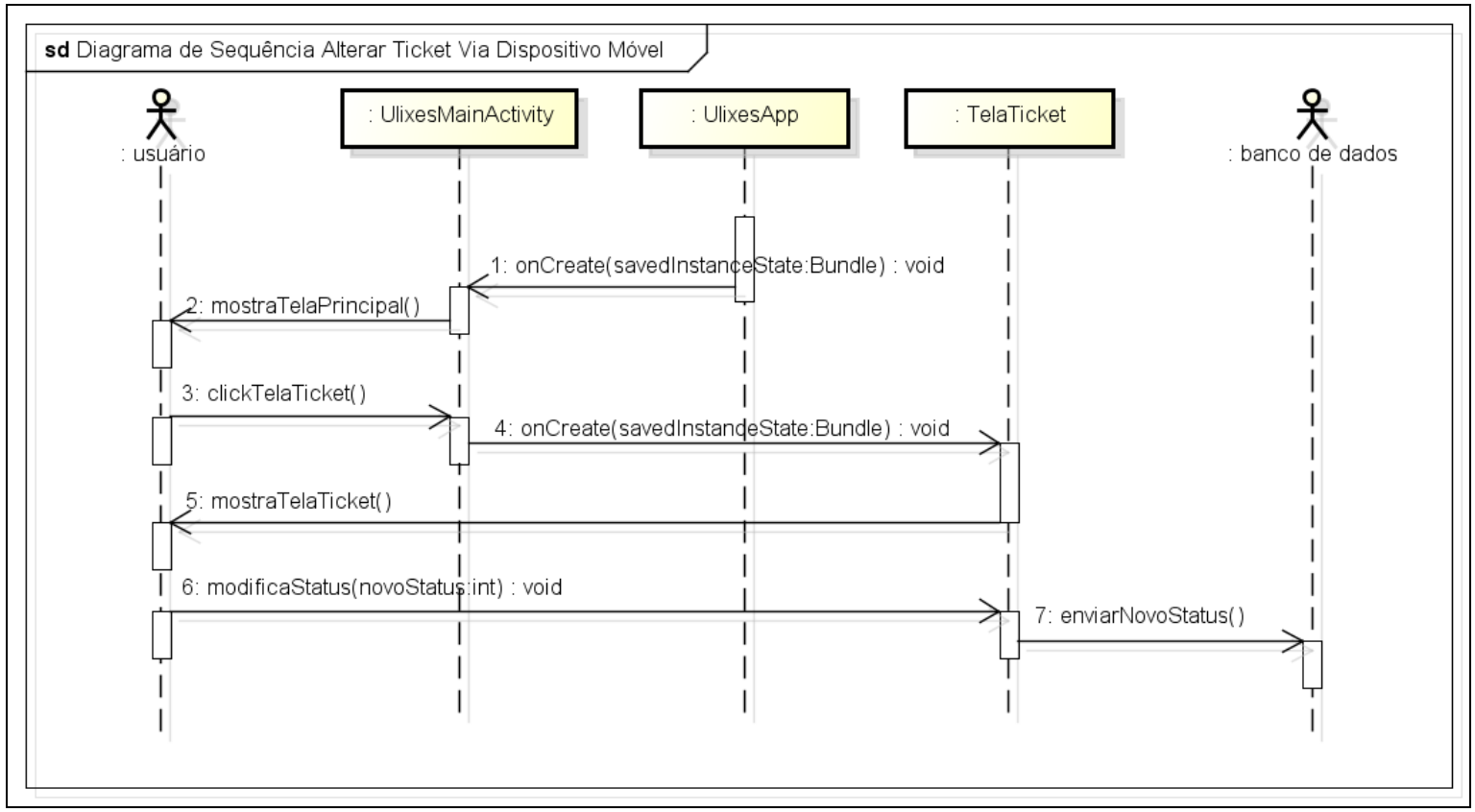


Figura 78 - Diagrama de Seqüência Alterar Ticket Via Dispositivo Móvel.

Fonte: Autoria própria