FACULDADE DE TECNOLOGIA DE AMERICANA
Curso Superior de Tecnologia em Análise de Sistemas e Tecnologia da Informação

GABRIELLI DE OLIVEIRA

MODELAGEM EM UML2, PARA SISTEMA COMERCIAL, EM EMPRESAS DE PEQUENO PORTE

Americana, SP
2014

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise de Sistemas e Tecnologia da Informação

GABRIELLI DE OLIVEIRA

MODELAGEM EM UML2, PARA SISTEMA COMERCIAL, EM EMPRESAS DE PEQUENO PORTE

Trabalho monográfico, desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise de Sistemas e Tecnologia da Informação da Fatec Americana, sob orientação do Prof. Graduado José William Pinto Gomes.

Área de concentração: Engenharia de Software

Americana, SP

2014

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS

Dados Internacionais de Catalogação-na-fonte

Oliveira, Gabrielli de

O47m Modelagem em UML2 para sistema comercial em empresas de pequeno porte. / Gabrielli de Oliveira. – Americana: 2014.

44f.

Monografia (Graduação de Tecnologia em Análise de Sistema e Tecnologia da Informação). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.

Orientador: Prof. José William Pinto Gomes

1. Desenvolvimento de software 2. UML – Linguagem de programação I. Gomes, José William Pinto II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.3.05

681.3.061

GABRIELLI DE OLIVEIRA

MODELAGEM EM UML2, PARA SISTEMA COMERCIAL, EM EMPRESAS DE PEQUENO PORTE

Trabalho de conclusão de curso apresentado à Faculdade de Tecnologia de Americana como parte dos requisitos para obtenção do título de Tecnólogo em Análise de Sistemas e Tecnologia da Informação.

Área de concentração: Engenharia de Software

Americana, 23 de junho de 2014 .

Banca Examinadora:

_____
José William Pinto Gomes
Graduado
Faculdade de Tecnologia de Americana

_____
Francesco Artur Perrotti
Mestre
Faculdade de Tecnologia de Americana

_____
Sisino Motta Neto
Especialista
Faculdade de Tecnologia de Americana

AGRADECIMENTOS

Primeiramente a Deus que permitiu que tantas coisas acontecessem, não somente na vida universitária, mas ao longo da minha vida, e que em todos os momentos me ajudou sendo o melhor mestre e Pai que já conheci.

À FATEC Americana pela oportunidade e também pelo bom ambiente que nos proporciona.

Ao meu orientador pelo empenho ao dedicado á orientação deste trabalho.

Agradeço aos meus pais Sidnei e Maria, pelo incentivo que me fortaleceu e me sustentou nas horas difíceis, de desânimo e cansaço, além de todo apoio dado em toda minha vida.

Agradeço ao meu irmão Felipe, por estar do meu lado em todos os momentos sendo mais que um irmão.

A todos que direta, ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

DEDICATÓRIA

Aos meus pais e irmão que, com muito carinho e apoio não mediram esforços para que eu chegasse até esta etapa de minha vida.

RESUMO

Esta monografia apresenta um breve estudo sobre Engenharia de *Software* e UML (*Unified Modeling Language* - Linguagem de Modelagem Unificada). O principal tema abordado será a importância e o uso da UML para a modelagem de sistemas para pequenas empresas durante o seu desenvolvimento, partindo do princípio de que durante o desenvolvimento de *software* para pequenas empresas a Engenharia de *software* muitas vezes não é utilizada. Serão descritos quais os diagramas contidos na UML e sua importância no processo, e, através de um estudo de caso será modelado um sistema onde somente os diagramas essenciais para o desenvolvimento de um *software* para uma EPP (Empresa de Pequeno Porte) serão utilizados de acordo com a necessidade do *software*.

Palavras-chave: Engenharia de *Software*; UML; Modelagem; Empresa de Pequeno Porte.

ABSTRACT

Modeling Language Unified - Modeling Language). The main issue addressed is the importance and use of the UML for modeling systems for small businesses during its development, assuming that during the development of software for small businesses software engineering is often not used. The diagrams contained in UML will be described with their importance in the process, and only the critical diagrams for developing software for a small business, through a case study, will be showed according to the real need of the software.

Keywords: *Software Engineering; UML; Modeling; Small Business.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CASE: *Computer-Aided Software Engineering*

OMG: Grupo de Gerenciamento de Objetos

UML: *Unified Modeling Language*

OMT: *Object Modeling Technique*

OOSE: *Object-Oriented Software Engineering*

SADs: Sistema de Apoio à Decisão

SAEs: Sistema de Apoio ao Executivo

SEBRAE: Serviço Brasileiro de Apoio a Micro e Pequenas Empresas

SGBD: Sistema Gerenciador de Banco de Dados

SIGs: Sistemas de Informações Gerenciais

SPTs: Sistemas de Processamento de Transações

SUMÁRIO

1 INTRODUÇÃO

O conceito de Engenharia de software foi inicialmente proposto em 1968, em uma conferência para discutir o que foi chamado de então ‘crise de software’ (SOMMERVILLE, 2007, p. 3), essa crise, dependia diretamente da introdução de um novo hardware de computador baseado em circuitos integrados. A experiência inicial na construção desses sistemas mostrou que o desenvolvimento informal de softwares não era eficiente, já que muitas vezes um projeto importante demorava anos para ser concluído, extrapolando assim custos e trazendo instabilidade em relação à segurança e desempenho, de fato o desenvolvimento de software estava em crise. Novas técnicas e métodos surgiram a fim de controlar a complexidade dos grandes sistemas de software, as quais são parte da engenharia de software e amplamente utilizadas hoje em dia. Segundo Guedes (2009, p.19), até meados da década de 1990, os três métodos mais conhecidos e populares eram o método de Booch, OMT (*Object Modeling Technique*) e o método OOSE (*Object-Oriented Software Engineering*) de Rumbaugh, e foi a partir da união dos três métodos mencionados que surgiu a UML (*Unified Modeling Language*) financiada pela Rational Software e sendo adotada como linguagem-padrão de modelagem pela OMG (*Object Management Group*). Em 2005, a versão 2.0 da linguagem foi lançada, e atualmente se encontra na versão 2.2.

Guedes (2009, p.19) explana que a UML não é uma linguagem de programação, e sim uma linguagem visual utilizada para modelar softwares que estão enquadrados dentro do paradigma de orientação a objetos, podendo ser aplicada a todos os domínios de aplicação, cuja finalidade é auxiliar os engenheiros de software a definirem os requisitos e características do software, seu comportamento, estrutura e até mesmo as condições físicas necessárias a esse software.

Muitas empresas de pequeno porte, não possuem um sistema para facilitar o gerenciamento de funções básicas. Esse problema ocorre por que as soluções existentes atualmente possuem um valor elevado e também pela falta de oferta de sistemas para pequenos negócios, obrigando então os administradores a utilizarem soluções simples, como anotações manuais, planilhas eletrônicas, a fim de controlar suas vendas e aquisições, por exemplo. Uma das dificuldades desse segmento está

na precariedade e na baixa consistência das informações, pois a maioria dos dados disponíveis é oriunda de fontes secundárias e apresentam lacunas significativas.

Justifica-se a escolha deste problema por que a autora do trabalho se interessa em desenvolver software para pequenos negócios utilizando o método UML.

O objetivo geral deste trabalho é fornecer uma documentação adequada e apresentar a modelagem de sistema em UML para uma empresa fictícia de pequeno porte.

Os objetivos específicos são:

- Fazer o levantamento bibliográfico sobre engenharia de software e o meio em que será aplicada;
- Estudar o método de software UML e a forma como será utilizada no desenvolvimento da modelagem do software para uma empresa de pequeno porte;
- Realizar um estudo de caso, utilizando o método proposto.

Os procedimentos metodológicos utilizados nesse trabalho serão de natureza exploratória e experimental, por meio de uma pesquisa bibliográfica através de livros e artigos científicos no aspecto teórico do trabalho, com fim descritivo sobre conceitos relacionados a engenharia de software e UML, e também de um estudo de caso aplicando os conhecimentos adquiridos (GIL, 2002).

O capítulo 2 apresenta o levantamento bibliográfico sobre Engenharia de software e UML, abordando primeiramente as fases de Levantamento e Análise de Requisitos, e descrição dos principais diagramas utilizados. O capítulo 3 detalha o estudo de caso. A análise e discussão dos resultados obtidos são feitas no capítulo 4. As considerações finais no capítulo 5.

2 LEVANTAMENTO BIBLIOGRÁFICO

2.1 Engenharia de software: uma visão geral

Fritz Baur, citado por Pressman (1995, p.31), define engenharia de software como “estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquina reais”. Está relacionada a todos os aspectos de produção de software, desde os estágios iniciais até a sua manutenção e “procura selecionar o método mais apropriado para um conjunto de circunstancias” (SOMMERVILLE, 2007, p. 5), estando fundamentada em um comprometimento organizacional com a qualidade.

Pressman afirma que a engenharia de software abrange um três elementos fundamentais – métodos, ferramentas e procedimentos, descritos a seguir.

Os métodos fornecem informações técnicas para que o software possa ser desenvolvido, e envolve diversas tarefas de modelagem ou descritivas, como: comunicação, análise de requisitos, modelagem de projeto, construção do programa, testes e suporte, “muitas vezes introduzem uma notação gráfica ou orientada a uma linguagem especial e introduzem um conjunto de critérios para a qualidade do software” (PRESSMAN, 1995, p. 31). Segundo Sommerville (2007, p. 8), não existe um método ideal, e diferentes métodos possuem diferentes áreas onde são mais aplicáveis, como por exemplo, para sistemas interativos, o método orientado a objeto é o mais aplicável, o que não funcionaria para sistemas com requisitos de tempo real.

“As ferramentas de engenharia de software proporcionam apoio automatizado (ou semi-automatizado) aos métodos” (PRESSMAN, 1995, p.32). Todos os métodos possuem uma ferramenta CASE (*Computer-Aided Software Engineering*) associada como, por exemplo, editores para notações utilizadas nos métodos, módulos de análise, geradores de relatório para a criação da documentação do sistema (SOMMERVILLE, 2007, p.9).

Os procedimentos se constituem no elo que une os métodos e as ferramentas, ele define a ordem em que os métodos serão aplicados, quais os produtos que precisam ser entregues (relatórios, documentos etc.), assegurando assim a qualidade da entrega.

2.1.1 Processo de software

"Um processo de software é um conjunto de atividades e resultados associados que produz de um produto de software" (SOMMERVILLE, 2007, p.6), porém, segundo Pressman (1995), os processos não definem de maneira rigorosa, como uma prescrição, os passos para o desenvolvimento do software, mas sim, possibilita às pessoas (equipe de desenvolvimento) que escolham o conjunto apropriado de ações e tarefas para que o desenvolvimento ocorra, seja entregue no prazo estipulado e com qualidade.

Segundo Pressman (2011, p.39), a engenharia de software é uma tecnologia em camadas, onde a camada de processo é sua base. "O processo de engenharia de software é a liga que mantém as camadas de tecnologia coesas e possibilita o desenvolvimento de software de maneira racional e dentro do prazo" (PRESSMAN, 2011, p.39) e é através do processo que é definida a metodologia para a entrega da tecnologia da engenharia de software.

> "O processo de software constitui a base para o controle do controle para o gerenciamento de projetos de software e estabelece o contexto no qual são aplicados métodos técnicos, são produzidos produtos derivados (modelos, documentos, dados, relatórios, formulários etc.), são estabelecidos marcos, a qualidade é garantida e mudanças são geridas de forma apropriada." (PRESSMAN, 2011, p.40).

Existem muitos processos de softwares diferentes, Pressman (1995, p. 46) afirma que a fase de desenvolvimento possui apenas três fases, independente de sua área de aplicação, tamanho do projeto ou complexidade que são definição, desenvolvimento e manutenção.

A fase de definição tem como foco o que, assim o desenvolvedor precisa saber as informações que serão processadas, as funções do software e desempenho esperado, suas restrições, requisitos, interfaces e critérios para validação. Nessa fase também são identificadas as exigências fundamentais do sistema e do software. Dentro dessa fase outras três etapas ocorrerão:

a) Análise do Sistema: é onde será definido o papel que o software terá, tendo as funções do sistema descritas.
b) Planejamento de projeto de software: após a definição dos requisitos, os riscos serão analisados, os recursos alocados, e os custos serão estimados para que o prazo seja atendido.

c) Análise de Requisito: para que o software seja desenvolvido é necessário que exista um domínio das informações e funções que serão necessárias ao software. Nessa fase é realizada a análise detalhadas dos requisitos necessários.

A fase de desenvolvimento focaliza como, ou seja, como o desenvolvedor irá desenvolver o software, qual será a estrutura de dados, arquitetura projetada, os detalhes do mesmo, qual será a linguagem de programação utilizada assim como também será definido a maneira que esse software será testado. Os métodos aplicados podem ter variações, mas sempre possuirá os seguintes passos, que segundo Pressman (1995, p.47) são:

a) Projeto de Software: Onde os requisitos de software serão traduzidos de maneira gráfica descrevendo a estrutura de dados, arquitetura, procedimento algorítmico, e as características de interface.
b) Codificação: Depois do projeto de software ter sido feito, é iniciada a codificação em alguma linguagem de programação, os programadores irão implementar os requisitos de acordo com a necessidade do sistema.
c) Realização de testes do software: Após o software ser implementado de maneira executável, o software precisa ser testado para que seus defeitos de função, lógica e implementação possam ser identificados.

A fase de manutenção está concentrada na fase de mudanças que está associada à correção de erros, adaptações que serão exigidas à medida que o software evolua e ampliações requisitadas pelo cliente. Três tipos de mudanças são encontrados durante a fase de manutenção:

a) Correção: É provável que o cliente encontre defeitos no software, nesse caso a manutenção corretiva é necessária para mudar o software e corrigir os defeitos encontrados.
b) Adaptação: Algum tempo depois da implementação do software, o ambiente de trabalho pode sofrer alterações, nesse caso a manutenção adaptativa, modificará o software para que ele continue funcionando no ambiente atual.
c) Melhoramento funcional: Após começar a utilizar o software o cliente poderá necessitar de funções adicionais. A manutenção perfectiva faz com que o software atenda suas novas exigências funcionais.

2.1.2 Modelagem de software

Segundo Pressman, para que um software seja desenvolvido com sucesso, é fundamental que o software seja planejado e possua um roteiro de construção com o melhor caminho para o software pronto. A modelagem abrange tanto a fase de análise do software quanto o projeto do mesmo, possuindo o objetivo de formalizar a compreensão do trabalho que será feito, além de prover embasamento técnico aos desenvolvedores.

"Criam se modelos para uma melhor compreensão do que será realmente construído" (PRESSMAN, 2011, p.116). Ele também explica que quando algo físico é construído, como por exemplo, um prédio, seu modelo é exatamente igual a sua futura construção física, porém em escala menor, porém quando esse conceito é levado ao desenvolvimento de um software, isso não se torna possível, por esse motivo a modelagem de um software assume uma forma diferente. O modelo de um software representa as informações as quais ele transforma, sua arquitetura, além de funções e características desejadas pelo usuário, iniciando-se no nível de compreensão do cliente e posteriormente, a um nível mais técnico.

Pressman (2011, p. 39) explica que existem duas classes de modelagem na engenharia de software, as quais são modelos de requisitos e modelos de projeto. Os modelos de requisitos definem o software da maneira que o usuário o requisitou, descrevendo suas funções, seu comportamento desejado e informações manipuladas. Os modelos de projeto representam as características que ajudarão os desenvolvedores na construção do software, como a arquitetura, sua interface gráfica para o usuário, e diversos detalhes quanto aos seus componentes.

2.2 Empresa de Pequeno Porte

Ainda não existe uma unanimidade para a classificação de uma empresa de pequeno porte. Segundo a legislação, lei complementar número 123, de 14 de dezembro de 2006, uma empresa de pequeno porte consiste na "sociedade empresária, a sociedade simples, a empresa individual de responsabilidade limitada e o empresário" que esteja registrado no Registro de Empresas Mercantis ou no Registro Civil de Pessoas Jurídicas, e que "aufira, em cada ano-calendário, receita bruta superior a R$ 360.000,00 (trezentos e sessenta mil reais) e igual ou inferior a R$ 3.600.000,00 (três milhões e seiscentos mil reais)".

Outra maneira de se classificar o tamanho de uma empresa é pelo número de seus funcionários, o SEBRAE (Serviço Brasileiro de Apoio a Micro e Pequenas Empresas) além de utilizar os critérios da Lei Complementar 123/2006, também considera uma empresa como sendo de pequeno porte, caso a mesma possua entre 20 a 99 empregados, caso seja do ramo Industrial, ou 10 a 49 empregados caso seja do ramo de Comércio e Serviços.

2.3 Sistema de informação comercial

Antes de tudo é importante definir o conceito do termo sistema, que segundo O'Brien (2004, p.7) "é um grupo de componentes inter-relacionados que trabalham juntos rumo a meta comum recebendo insumos e produzindo resultados em um processo organizado de transformação". Stair e Reynolds (2012, p. 7) destacam que os sistemas possuem elementos que relacionados entre si determinam como esse sistema funciona.

Laudon e Laudon (2011) definem Sistemas de Informação (SI), como um conjunto de componentes que se relacionam entre si para coletar (ou recuperar), processar, armazenar e distribuir informações tendo como objetivo auxiliar as tomadas de decisões, coordenação e controle das organizações, além de auxiliar os trabalhadores a "analisar problemas, visualizar assuntos complexos e criar novos produtos".

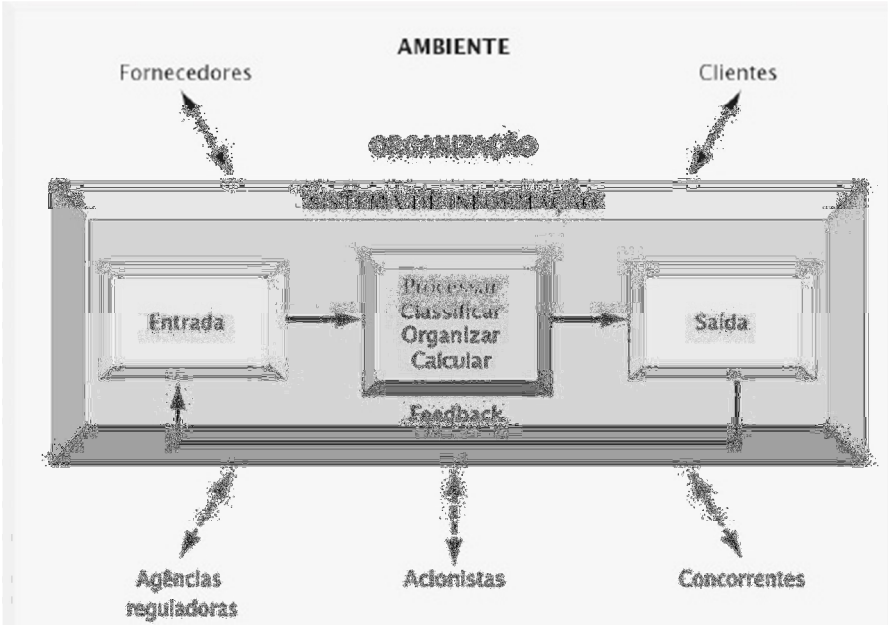
Os sistemas de informação possuem informações sobre pessoas, locais ou itens que possam ser significativos para a empresa e seu ambiente. Nesse caso, dados são os fatos crus (STAIR; REYNOLDS, 2012, p.4) que ainda não foram analisados de maneira que possam ser entendidos e arranjados de acordo com a melhor maneira, as informações são dados apresentados de maneira útil e significativa para os seres humanos (LAUDON; LAUDON, 2011, p.12).

Ainda segundo o mesmo autor, para que as organizações possam tomar suas decisões, controlas suas operações, realizar analise de problemas e também criar novos produtos, um sistema de informação é dividido em três atividades que são entrada, processamento e saída, descritos a seguir:

a) Entrada: consiste na coleta dados brutos dentro de uma organização, que serão necessários;
b) Processamento: converte os dados brutos capturados na entrada, em resultados úteis e significativos;

c) Saída: normalmente em formato de documentos e relatórios (Stair e Reynolds, 2012, p.10), transfere as informações ás atividades que serão empregadas, ou as pessoas que a utilizarão.

Figura 1 - Sistema de Informação



Fonte: Laudon e Laudon (2011)

2.3.1 O papel dos sistemas de informação em uma empresa

Segundo Laudon e Laudon (2011), os sistemas de informação ocupam um papel crítico dentro do ambiente organizacional, pois é a maneira que as empresas têm de para administrar suas funções internas, lidar com suas demandas entres outros. O mesmo autor ainda afirma que as empresas investem em sistemas de informação para poder atingir excelência operacional, desenvolver novos produtos, estreitar o relacionamento com seus clientes e poder atendê-los de uma maneira melhor, melhoras o embasamento para tomada de decisões, aumentar suas competitividade no mercado, e assegurar sua sobrevivência. Cada empresa pode optar por um tipo diferente de sistema para sua organização, “esses sistemas podem ser classificados em sistemas de processamento de transações, sistemas de informações gerenciais, sistemas de apoio à decisão e sistema de apoio ao executivo” (LAUDON; LAUDON, 2011).

a) Sistemas de processamento de transações (SPTs): são sistemas computadorizados que registram informações e realiza transações e atividades básicas da organização, além de serem fontes de informação para outros tipos de sistema. Em geral são críticos, pois afetam a empresa de maneira considerável caso enfrente problemas em seu funcionamento.

b) Sistemas de informações gerenciais (SIGs): são sistemas que atendem a gerentes de nível médio, fornecendo relatórios do desempenho atual da organização. Essas informações possibilitam que os gerentes possam controlar e monitorar a empresa, além de prever o seu desempenho futuro.

c) Sistemas de apoio à decisão (SADs): são sistemas que ajudam os gerentes de nível médio a tomar decisões que fogem da rotina, focam em problemas únicos que se alteram com rapidez, os quais não possuem resolução predefinida.

d) Sistema de apoio ao executivo (SAEs): ajudam a alta gerencia na tomada de decisões. São utilizados em situações não rotineiras, que por terem procedimentos pré-estabelecidos, contam com o bom senso e capacidade de avaliação desses gerentes.

2.4 Introdução a UML

"A UML – *Unified Modeling Language* ou Linguagem de Modelagem Unificada – é uma linguagem visual usada para modelar softwares baseados no paradigma de orientação a objetos" (GUEDES, 2009, p.19). Guedes explica que a mesma pode ser aplicada a todos os domínios de aplicação, e tornou-se a linguagem-padrão de modelagem internacionalmente de engenharia de software. A UML é uma linguagem de modelagem, e não de programação, com o objetivo de auxiliar os engenheiros de software a definirem as características do sistema, como seus requisitos, comportamento e estrutura lógica. Essas características podem ser definidas em UML antes mesmo de o software começar ser desenvolvido.

2.4.1 Levantamento e Análise de Requisitos

Segundo Guedes (2009, p. 21) uma das primeiras fases de um processo de desenvolvimento de software são as etapas de levantamento e análise de requisitos. Essas etapas trabalham com o domínio do problema e tem o objetivo de determinar o que o software deve fazer assim como também, se é possível realizar o desenvolvimento desse software da maneira exata como foi solicitado. Nessa etapa,

os dados são levantados pelos engenheiros através de entrevistas, a fim de entender melhor as necessidades dos usuários e o que ele deseja que o software realize.

Ainda segundo o mesmo autor, a fase de levantamento de dados deve identificar requisitos funcionais e não funcionais.

> "Requisitos funcionais correspondem ao que o cliente quer que o sistema realize as funcionalidades do software. Já os requisitos não funcionais são as restrições, condições, consistências e validações que devem ser consideradas em relação aos requisitos funcionais" (GUEDES, 2009, p.22).

Após a fase de levantamento de requisitos, inicia-se a fase em que todas as necessidades que o cliente apresentou serão analisadas, ou seja, a Análise de Requisitos. Nessa fase, o Engenheiro examina os requisitos apresentados pelos usuários, e verifica se eles foram especificados corretamente e bem compreendidos, de maneira que as questões sejam sanadas o mais breve possível.

2.4.2 Diagramas da UML

Guedes (2009, p. 30) explica que a UML é composta por múltiplos diagramas, com o objetivo de oferecer visões sobre o sistema a ser modelado. Cada diagrama analisa o sistema, ou parte dele, como se o sistema fosse modelado em camadas, alguns enfocam o sistema de forma geral, apresentando uma visão externa, enquanto outros dão um enfoque mais técnico, com uma visão mais profunda. Dessa maneira, falhas podem ser identificadas diminuindo assim, a possibilidade de erros.

O mesmo autor define os diagramas utilizados na UML:

a) Diagrama de Caso de Uso;

Embora o diagrama de caso seja utilizado principalmente na fase de levantamento e análise de requisitos, ele também pode servir de base para alguns outros diagramas, por esse motivo é o diagrama mais geral e informal da UML. O mesmo possui uma linguagem simples e identifica os atores que utilização o software, serviços e funcionalidades do sistema em questão.

b) Diagrama de Classes

O diagrama de classes serve de apoio aos demais diagramas e por isso é considerado o diagrama mais importante da UML. Ele define a estrutura das classes

que serão utilizadas pelo sistema, com seus atributos e métodos, além de estabelecer como as classes se relacionam e compartilham informações entre si.

c) Diagrama de Objetos

O diagrama de objetos é quase um complemento do diagrama de classes e também muito dependente deste, pois “fornece uma visão dos valores armazenados pelos objetos de um diagrama de classes em um determinado momento da execução de um processo de software” (GUEDES, 2009, p. 34).

d) Diagrama de pacotes

O diagrama de pacotes é um diagrama estrutural que representa os subsistemas ou submódulos de um sistema com o objetivo de determinar as partes que o compõem.

e) Diagrama de Sequência

O diagrama de sequência é um diagrama comportamental, ou seja, se preocupa com a ordem temporal em os objetos que estão envolvidos trocam mensagens entre si. É baseado no diagrama de caso de uso, e se apoia no diagrama de classes a fim de determinar os objetos das classes envolvidas.

f) Diagrama de Comunicação

Era conhecido como diagrama de colaboração até a versão 1.5 da UML, a partir da versão 2.0, teve seu nome modificado. Este diagrama está associado e é um complemento ao diagrama de sequência, pois as informações mostradas no diagrama de comunicação são frequentemente as mesmas mostradas no diagrama de sequências, mas com foco diferente, pois ele se concentra somente nos elementos do diagrama e não na temporalidade do processo.

g) Diagrama de Máquina de Estados;

O diagrama de máquina de estados “demonstra o comportamento de um elemento por meio de um conjunto finito de transições de estado, ou seja, uma maquina de estados” (GUEDES, 2009, p.37). É chamado como maquina de estado comportamental quando demonstra o comportamento de alguma parte do sistema, e

é chamado de máquina de estado de protocolo, quando expressa o protocolo doe uso de determinada parte do sistema.

h) Diagrama de Atividade

O diagrama de atividade descreve os passos que serão seguidos para que uma tarefa específica possa ser concluída, se se concentra na no fluxo de controle de uma atividade.

i) Diagrama de Visão Geral de Interação

“O diagrama de Visão Geral de Interação é uma variação do diagrama de atividade que fornece uma visão geral dentro de um sistema ou processo de negócio” (GUEDES, 2009, p. 39).

j) Diagrama de Componentes

O diagrama de componentes está fortemente relacionado a linguagem de programação que será utilizada no desenvolvimento do software, pois “representa os componentes do sistema quando o mesmo for ser implementado em termos de código-fonte, bibliotecas, formulários, arquivos de ajuda, módulos executáveis, etc.” (GUEDES, 2009, p.40 - 41) determinando a estrutura desses módulos e sua interação.

k) Diagrama de Implantação

O diagrama de implantação determina as necessidades do software em relação ao hardware do sistema e características físicas

l) Diagrama de Estrutura Composta

O diagrama de estrutura composta descreve a estrutura interna de uma classe ou componente, como se comunicam e colaboram entre si.

m) Diagrama de Tempo ou de Temporização

O diagrama de tempo descreve a mudança de estado de uma classe em resposta a eventos externos.

3 ESTUDO DE CASO

Neste capítulo, será modelado o sistema de informação para uma empresa de pequeno porte fictícia, usando-se como exemplo um teatro. O sistema será modelado por meio de quase todos os diagramas da UML, onde não será necessário considerar os diagramas de objetos, pacotes, visão geral de interação, implantação, estrutura composta, pois não se aplicam nessa situação. Os diagramas apresentados nesse estudo de caso não estão atrelados a nenhuma linguagem de programação específica e terão seu foco na funcionalidade de venda de ingresso do sistema.

3.1 Diagrama de Caso de uso

O caso de uso para o sistema de teatro, conforme Figura 2, demonstra o comportamento de suas funcionalidades na visão externa do usuário. É o diagrama mais informal, e é utilizado no início da modelagem do software, pois será consultado e utilizado como base para outros diagramas da UML.

No diagrama de caso de uso, foram identificados dois atores sendo Funcionário e Cliente.

- Funcionário: representa os funcionários que trabalham no teatro e tem a função de realizar a venda de ingressos, e manter os cadastros atualizados.

- Cliente: representa as pessoas que assistirão às peças que serão apresentadas nesse teatro.

Os casos de uso são:

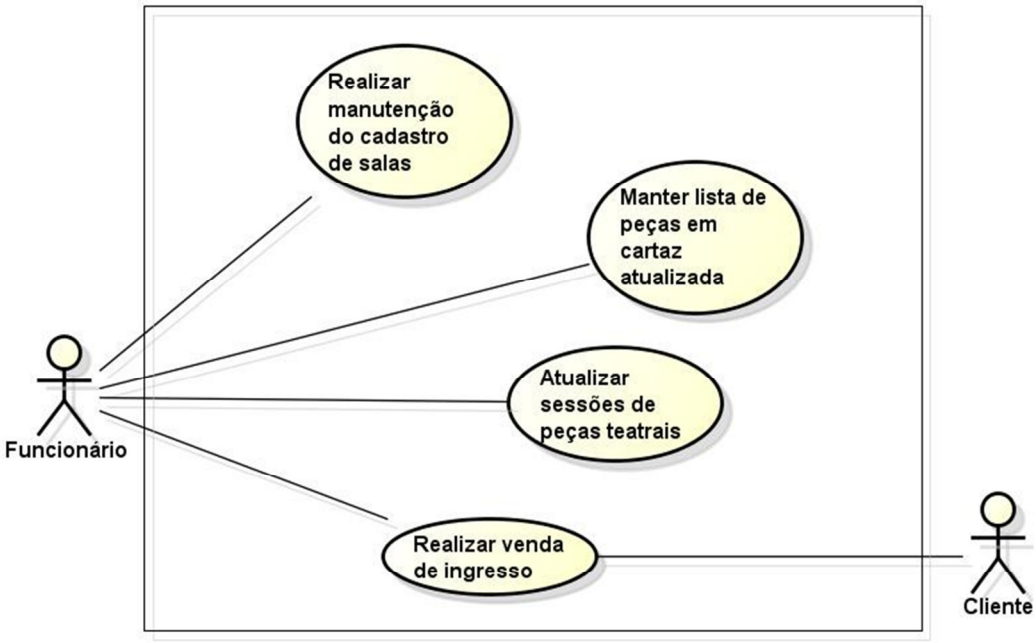
- Realizar manutenção do cadastro de salas: refere-se à manutenção das salas de teatro cadastradas no sistema.

- Manter lista de peças de teatro em cartaz atualizadas: refere-se ao processo de atualização das peças que atualmente estão em cartaz nesse teatro.

- Atualizar sessões de peças teatrais: representa a manutenção do cadastro de peças juntamente com sua sala e horários definidos.

- Realizar venda de ingresso: refere-se ao processo de venda de ingresso de uma sessão para o cliente.

Figura 2 - Diagrama de Caso de Uso - Sistema de Controle de Teatro



Fonte: Próprio autor.

Tabela 1 - Documentação do Caso de Uso Realizar Venda de Ingresso

Nome do Caso de Uso	Realizar venda de ingresso
Caso de Uso Geral	
Ator Principal	Funcionário
Atores Secundários	Cliente
Resumo	Este caso de uso descreve o caminho percorrido por um funcionário para a emissão de um ingresso para peça teatral
Pré-Condições	
Pós-Condições	
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Selecionar opção venda de Ingresso	
	2. Apresentar sessões disponíveis
3. Informar sessão desejada	
	4. Emitir ingresso

Restrições/ Validações	

Fonte: Próprio autor.

3.2 Diagrama de Classe

O diagrama de classes, demonstrado na Figura 3, é composto por suas classes e suas associações, tem seu foco principal em expor as classes que pelas quais o sistema será composto e mostrar seus relacionamentos entre si, mostrando também sua estrutura lógica. Esse diagrama também é usado como base para outros diagramas da UML.

- Gênero: a classe Genero armazena a descrição do gênero da peça apresentada pelo teatro.

- Peça_Teatral: essa classe contém o nome da peça e o tempo de duração da mesma. Uma peça só possui um gênero, mas um mesmo gênero pode possuir muitas peças.

- Ator: essa classe exibe os atores que interpretam os papéis da peça em questão.

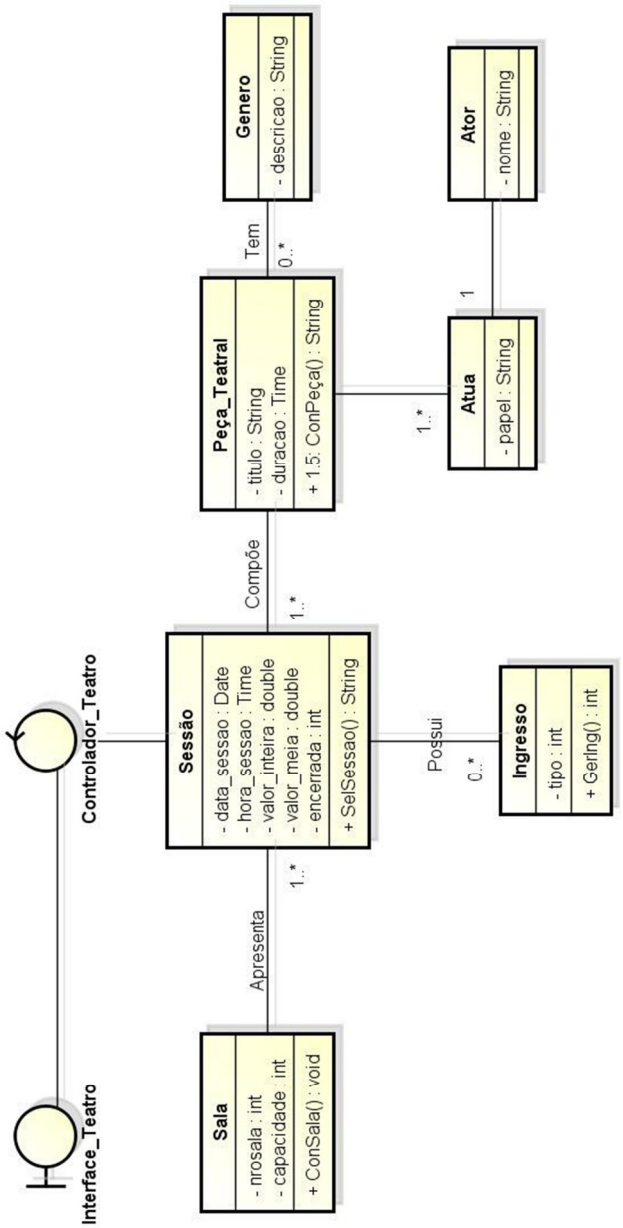
- Atua: classe intermediária entre Ator e Peça_Teatral, pois demonstra os papéis que cada ator interpreta em cada peça. Um ator pode atuar em uma única peça, porém uma peça pode ter no mínimo um e no máximo muitos atores.

- Sala: essa classe armazena as salas que o teatro possui, e seus atributos são número da sala e sua capacidade, tendo um único método para realizar essa consulta.

- Sessão: essa classe representa as sessões de peça apresentadas por esse teatro. Seus atributos são data, horário, valor do ingresso (inteira ou meia), e também um atributo para dizer se a sessão está ou não encerrada. Uma sessão é apresentada em uma única sala, porém uma única sala pode ter diversas sessões associadas a ela. Cada sessão pode ter somente uma peça sendo apresentada. O único método dessa classe seleciona as sessões ainda em aberto e retorna os dados dessa sessão.

- Ingresso: essa classe representa os ingressos vendidos para cada sessão, mostrando o tipo do mesmo (meia ou inteira), e também possui um método para gerar um novo ingresso.

Figura 3 - Diagrama de Classes para o Sistema de Controle de Teatro



Fonte: Próprio autor.

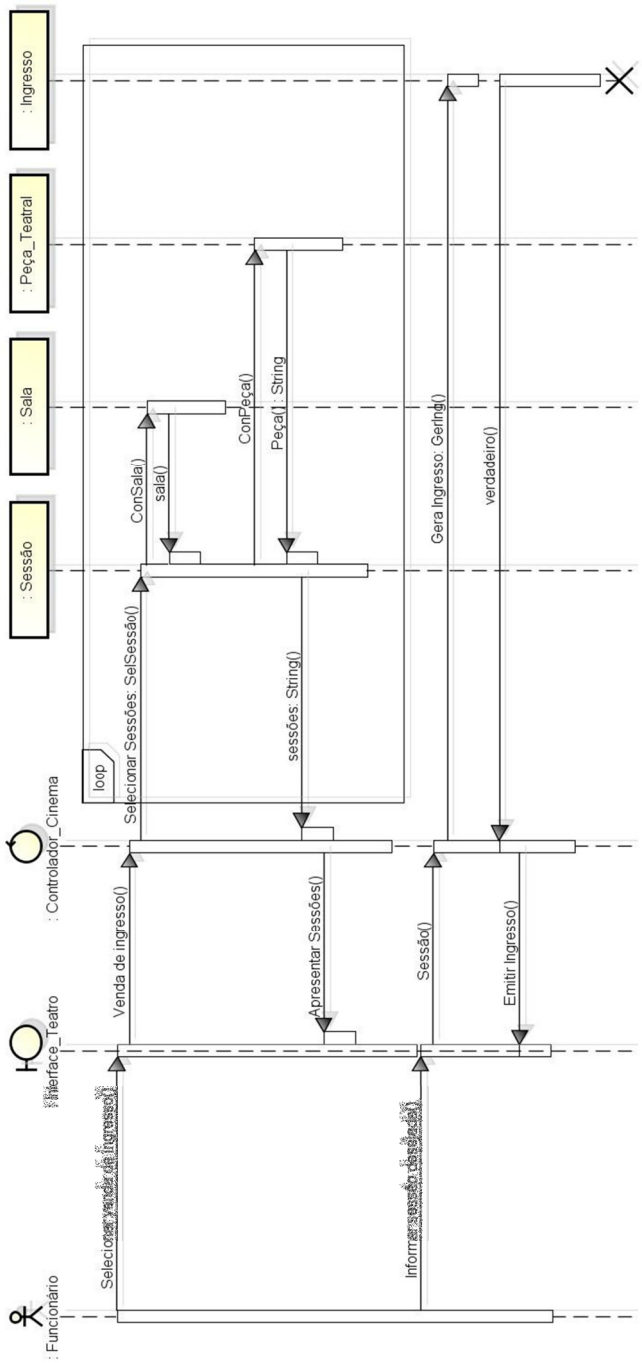
3.3 Diagrama de Sequência

O diagrama de sequência, conforme Figura 4, demonstra a sequência de eventos de determinado processo, assim como as mensagens que serão disparadas e sua ordem.

Quando a opção para realizar venda de ingresso for selecionada, o sistema carregará todas as sessões que ainda não terminaram, com o horário, peça e

número da sala que será apresentada. Após a escolha do cliente, o funcionário irá gerar o ingresso para a sessão escolhida.

Figura 4 – Diagrama de Sequência - Processo de Venda de Ingresso



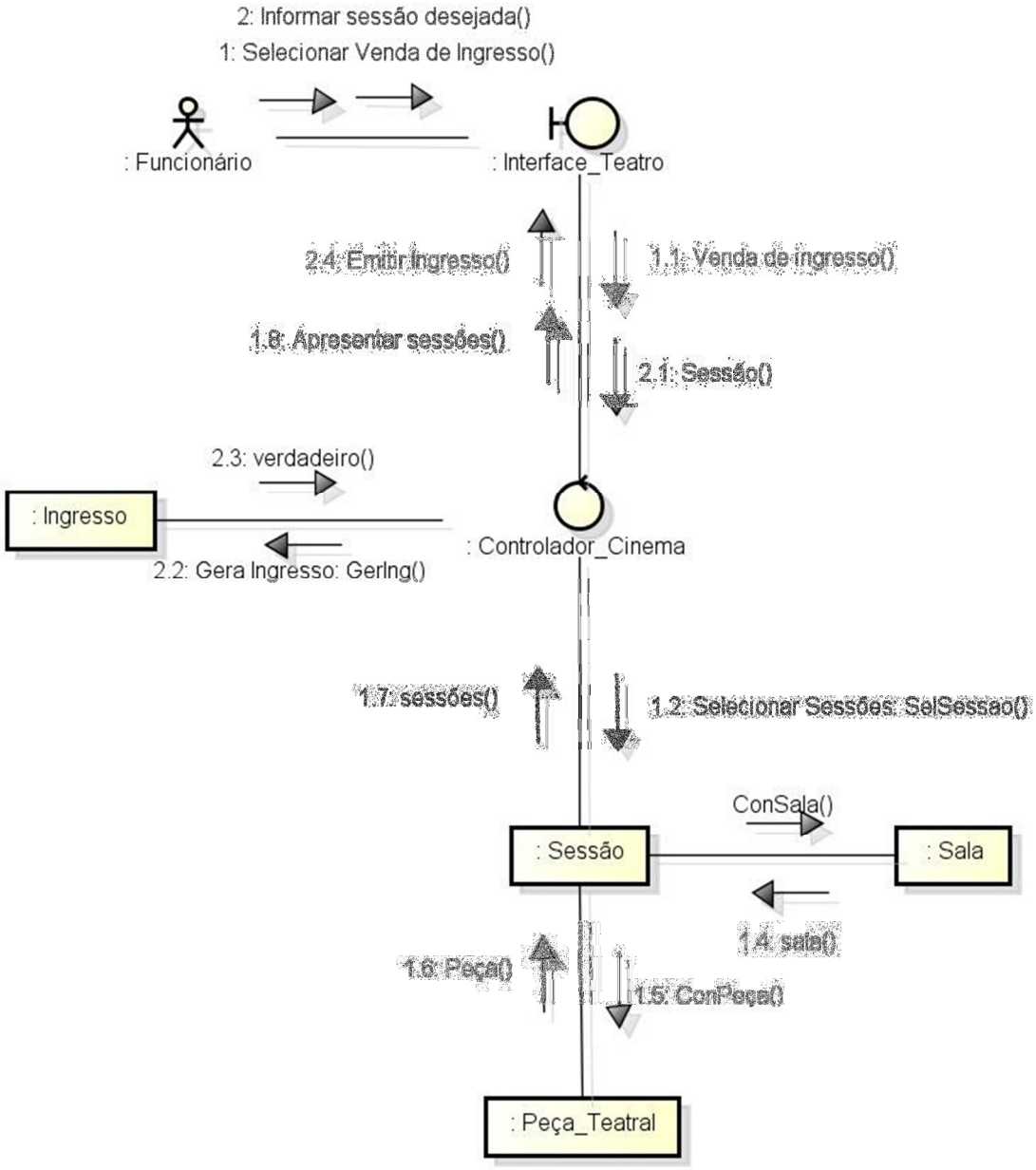
Fonte: Próprio autor.

3.4 Diagrama de Comunicação

O diagrama de comunicação, demonstrado na Figura 5, está relacionado ao diagrama de sequência, nesse estudo de caso suas informações são as mesmas,

porém com um foco diferente, pois esse diagrama não se preocupa com a temporalidade do processo e sim com as mensagens que são trocadas entre si.

Figura 5 - Diagrama de Comunicação - Processo de Venda de Ingresso



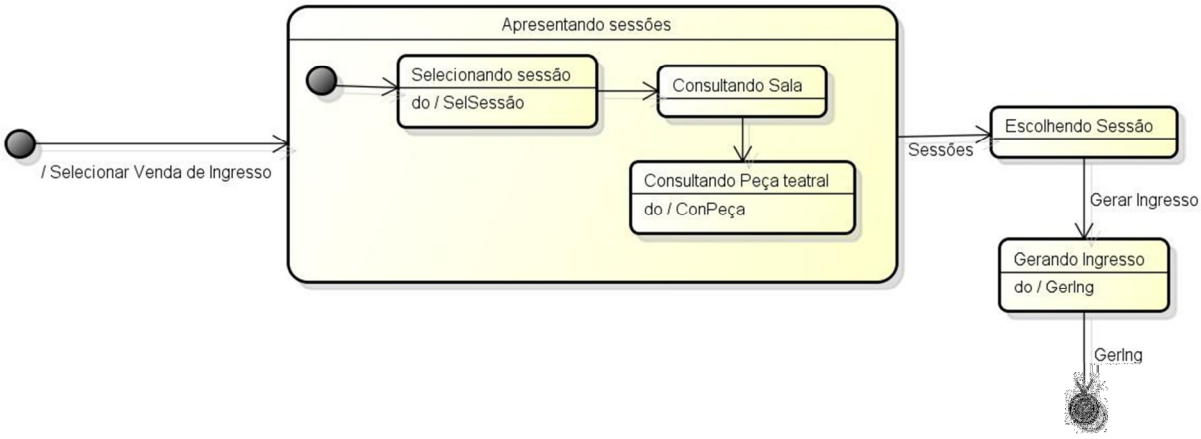
Fonte: Próprio autor.

3.5 Diagrama de Máquina de Estados

O diagrama de máquina de estados demostra o comportamento do sistema por meio de transições entre situações que o sistema se encontrará.

Nesse estudo de caso, no momento em que o funcionário selecionar a opção de venda de ingressos, o sistema retornará todas as sessões que não foram encerradas, contendo o nome da peça teatral e a sala em que será apresentada. A partir dessa lista, o funcionário selecionará a opção escolhido pelo cliente, e finalmente emitirá o ingresso para essa sessão.

Figura 6 - Diagrama de Máquina de Estados - Processo de Venda de Ingresso



Fonte: Próprio autor.

3.6 Diagrama de Atividade

O diagrama de atividade, vide Figura 7, apresenta muitas semelhanças com os antigos fluxogramas, pois enfatizam a lógica de programação para que se possa determinar um fluxo de controle.

3.7 Diagrama de Componentes

O diagrama de componentes, conforme Figura 8, demonstra os componentes que fazem parte do sistema, podendo ser lógicos ou físicos.

Nesse estudo de caso, o diagrama possui os seguintes componentes:

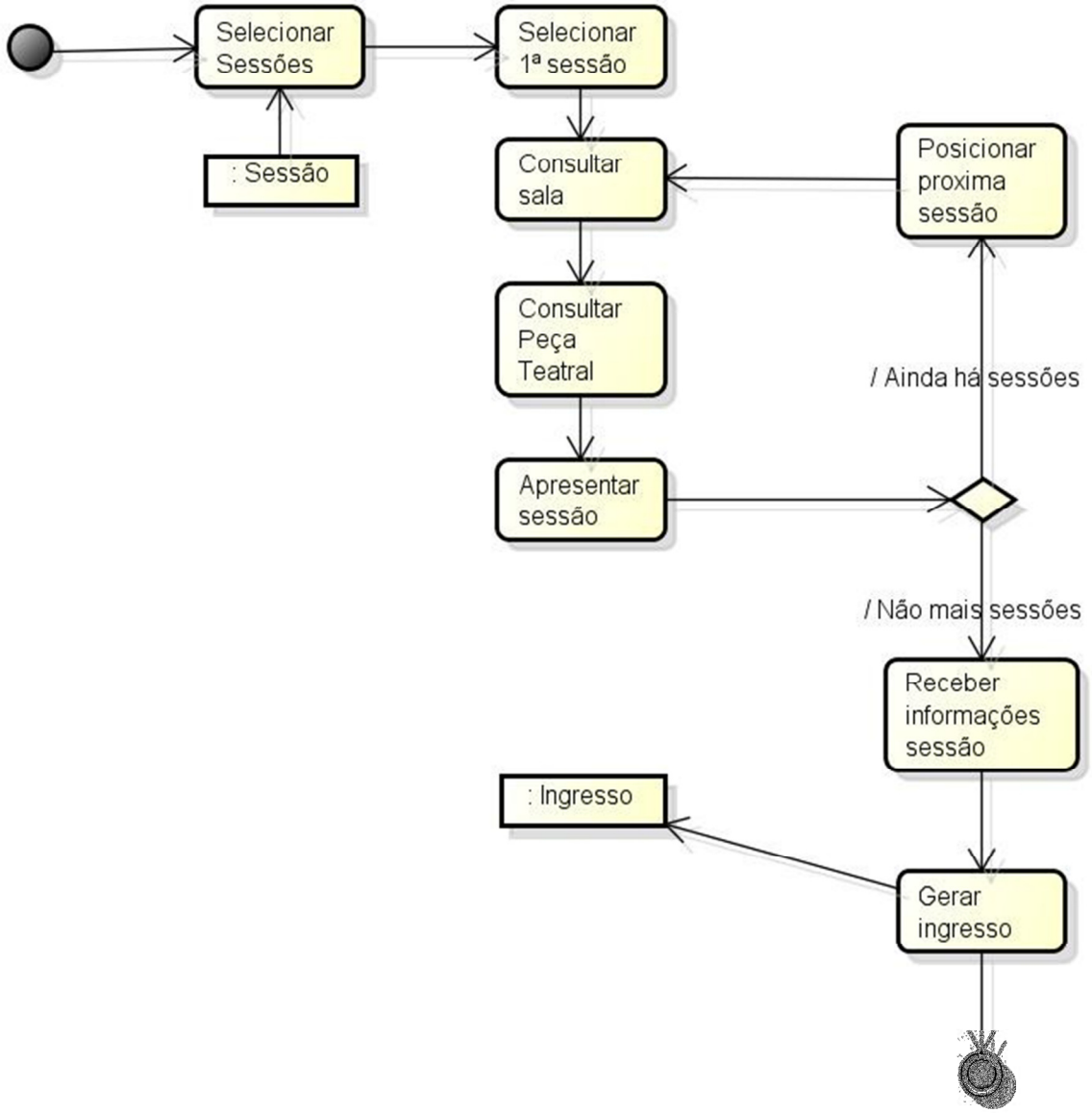
- Interface de venda de ingressos: representa não somente a interface do sistema, mas também o hardware que será necessário para a impressão dos ingressos.

- Módulo de venda de ingressos: gerencia o processo de venda.

- Sistema Gerenciador de Banco de Dados (SGBD) para manter as informações do sistema

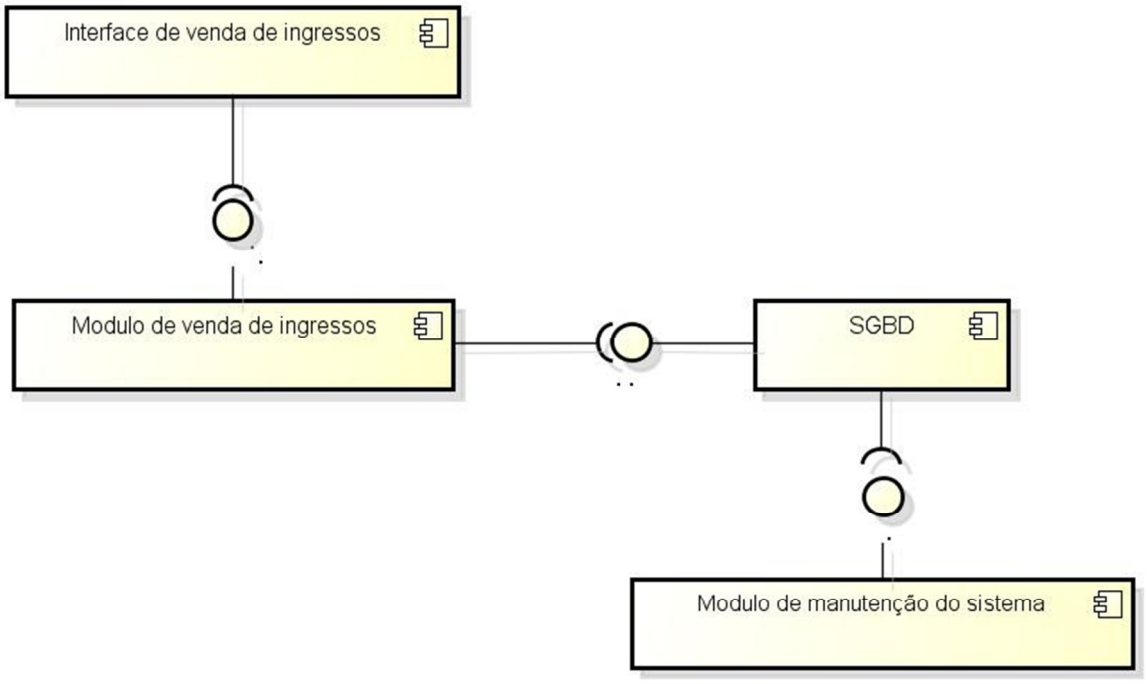
- Módulo de manutenção do sistema: responsável pela manutenção dos cadastros do sistema, como cadastro de salas, sessões, peças teatrais, gêneros, etc.

Figura 7 - Diagrama de Atividade - Processo de Venda de Ingressos



Fonte: Próprio autor.

Figura 8 - Diagrama de Componentes - Sistema de Controle de Teatro



Fonte: Próprio autor.

4 UTILIZAÇÃO DA UML

4.1 A importância do planejamento e o uso da UML

Segundo Sommerville (2007, p.63), o planejamento do software consiste em fazer o planejamento minucioso do progresso do software. Esse planejamento é feito de maneira interativa, pois será concluído somente quando o próprio projeto chega ao fim, pois à medida que as informações vão se tornando disponíveis, o plano precisa ser revisado e atualizado. O autor explica que planejar é necessário, pois até mesmo as restrições devem ser avaliadas, tais como data de entrega, pessoal que estará disponível, orçamento, etc., devido ao fato de estas afetarem, diretamente o desenvolvimento do projeto.

A UML é uma linguagem utilizada no processo de desenvolvimento, que pode ser empregada para a visualização, especificação, construção e documentação de artefatos que usem sistemas de software. Segundo os autores Booch, Rumbaugh e Jacobson (2005, p.13) ela é ideal para sistemas corporativos que podem ser distribuídos a aplicações Web.

A linguagem da UML é voltada para a representação de um sistema de maneira conceitual e física, com seus vocabulários e regras específicas ela é uma linguagem-padrão, para a construção de estruturas de projetos de software. Os mesmo autores ainda explicam que a modelagem de um software é muito importante para que se possa compreender um sistema, porém nenhum modelo será completamente suficiente, sendo por muitas vezes necessário utilizar diversos modelos conectados entre si para que a estrutura do software seja compreendida em sua totalidade. Para determinadas ocasiões, poderá ser mais adequado o uso de uma modelagem textual, para outras, uma modelagem gráfica, porém em ambos os casos, mostra-se a necessidade de uma modelagem para que o software que será programado futuramente seja representado de maneira que se possa compreender sua estrutura. Além de ser uma linguagem para visualização, a UML também é uma linguagem para especificação, pois através dela é construído modelos precisos, sem ambiguidades e completos capazes de atender as decisões importantes em termos de análise, projeto e implementação.

Guedes (2009, p.20) explica que além da necessidade primária de se desenvolver um projeto muito bem elaborado, é necessário fazer uma estimativa de

custos, determinando o tempo em que o projeto será concluído, quantidade de profissionais que serão alocados, além do material necessário, então o autor conclui que todos os sistemas devem ser modelados antes do início de sua implementação, por menores e mais simples que sejam, pois frequentemente aumentam de tamanho, complexidade e abrangência. Os clientes podem necessitar de modificações e melhorias, novas leis são criadas, e o modelo de mercado ao qual o sistema esta inserido pode mudar, por esses motivos, é preciso um sistema bem documentado para que possa ser mantido com maior facilidade, isto é, suas correções e atualizações não produzam novos erros.

4.2 Diagramas da UML

Para Guedes (2009, p.30) a UML é formada por diversos diagramas devido à necessidade de o sistema possuir múltiplas visões através da modelagem, para que assim seus aspectos possam ser analisados. Cada diagrama da UML apresenta o sistema de um determinado ponto de vista, como se o sistema fosse modelado em camadas, porém com enfoques diferentes (técnicos, profundos, geral, etc.). O uso dos diagramas é muito importante para que seja possível identificar possíveis falhas assim como diminuir a ocorrência de erros futuros.

4.2.1 Diagrama de caso de uso

Booch, Rumbaugh e Jacobson (2005, p. 227) explicam que os sistemas precisam interagir entre si com atores humanos, que utilizarão os sistemas para determinados propósitos, dessa maneira, os atores desejam que o sistema de comporte da maneira m que foi prevista. Como é demonstrado na Figura 9, o caso de uso discrimina o comportamento de um sistema (ou parte dele) descrevendo um conjunto sequências de ações realizadas pelo sistema, podem ser aplicados de maneira que captem o comportamento desejado do sistema, porém sem a necessidade de descrever como esse comportamento será implementado pelos desenvolvedores, permitindo assim, que tanto o usuário como o especialista, focalizem nos postos de maior risco. Além disso, também contribuem para que os desenvolvedores consigam ter uma visão completa do sistema e compreendam que é desejado pelo usuário final.

“Casos de uso bem estruturados denotam somente o comportamento essencial do sistema ou subsistema e não amplamente gerais, nem muito específicos” (BOOCH; RUMBAUGH; JACOBSON, 2005, p.227).

No nível do sistema, os casos de uso descrevem o conjunto de sequências que representam interação com recursos externos ao sistema (atores) e também com seu próprio sistema (abstrações).

Booch, Rumbaugh e Jacobson (2005, p.238) consideram a aplicação dos casos de uso importantes por três razões:

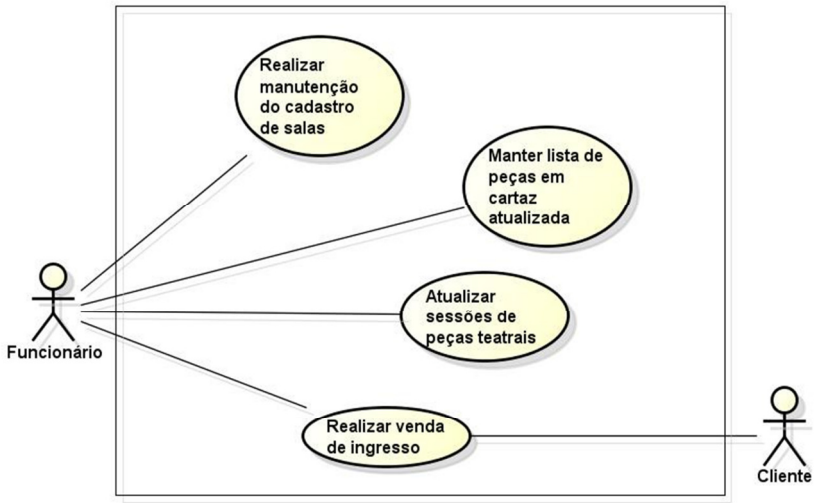
1) A partir da modelagem dos elementos através do caso de uso, os especialistas do domínio adquirem uma visão externa suficiente para que os desenvolvedores tenham informações suficientes para realizar a construção interna.

2) O caso de uso permite que o autor do caso de uso, exponha a sua intenção de como determinado elemento será utilizado, pois caso contrário, os usuários teriam que descobrir sozinhos como usar cada elemento.

3) Os casos de uso servem como base para a realização de testes dos elementos, o que faz com que o software possa ser testado continuamente.

Assim, os diagramas de caso de uso são importantes para visualizar, especificar e documentar o comportamento de um elemento, fazendo com que os sistemas fiquem compreensíveis devido a sua apresentação externa de como serão utilizados no contexto, a fim de que o desenvolvedor possa implementá-lo.

Figura 9 - Exemplo de diagrama de caso de uso



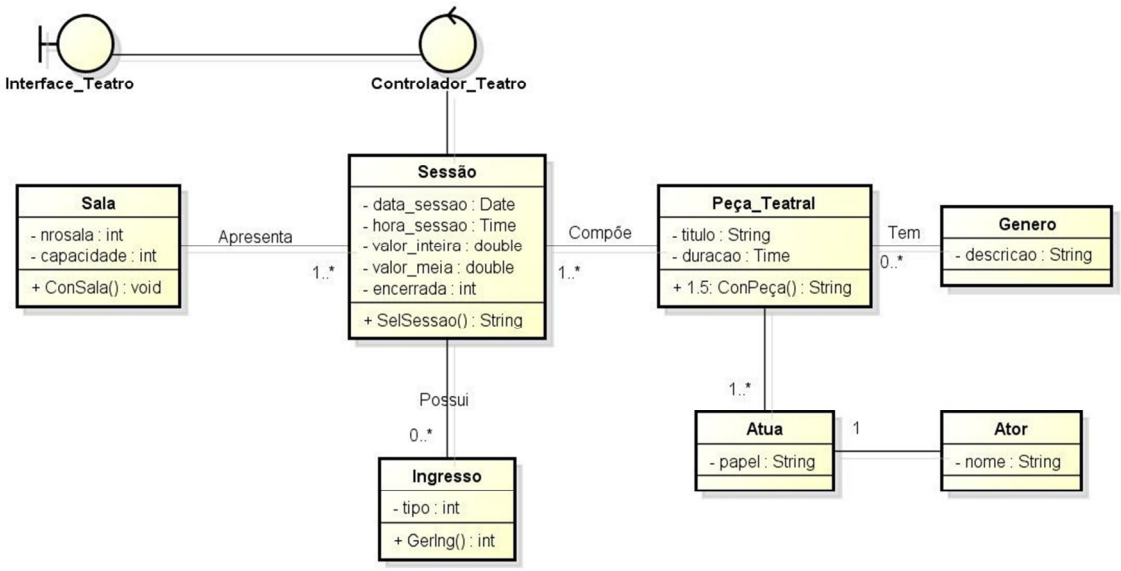
Fonte: Próprio autor

4.2.2 Diagrama de classes

Segundo Booch, Rumbaugh e Jacobson (2005) as classes são os blocos de construção mais importantes de qualquer sistema, pois descrevem o conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica.

O diagrama de classes, de acordo com a Figura 10, apresenta uma imagem de como as classes serão organizadas, preocupando-se com a estrutura lógica delas. O diagrama de classes servirá de base para diversos outros diagramas da UML.

Figura 10 - Exemplo de diagrama de classes



Fonte: Próprio autor

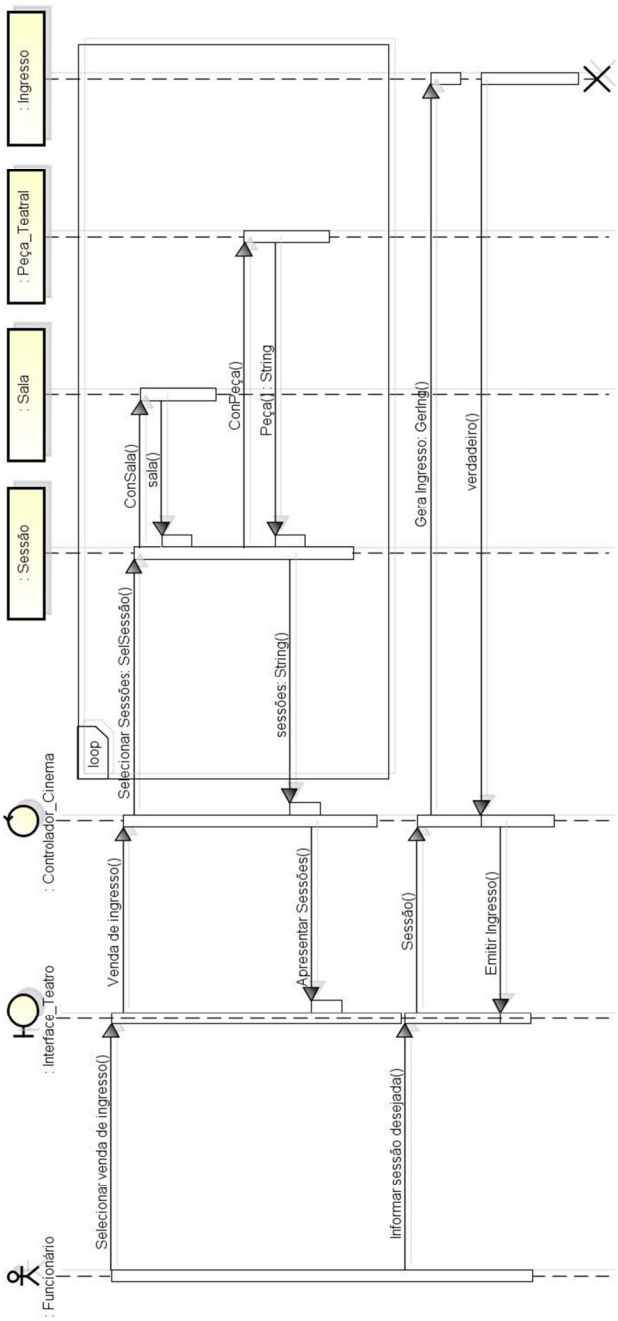
4.2.3 Diagrama de sequência

O diagrama de sequência é um diagrama comportamental, baseado no diagrama de caso de uso, dessa maneira, para cada caso de uso, normalmente exista um diagrama de sequência descrevendo o processo disparado pelo ator.

O diagrama de sequência também se baseia no diagrama de classes, que será a fonte das classes que serão utilizadas.

Sua principal função é expor o que acontece em cada processo, como se pode ver na Figura 11, apontando as mensagens disparadas pelos atores e a ordem em que isso acontecerá, mensagens enviadas, métodos chamados e os objetos que interagem entre si. (GUEDES, 2009, p. 200).

Figura 11 - Exemplo de diagrama de sequência



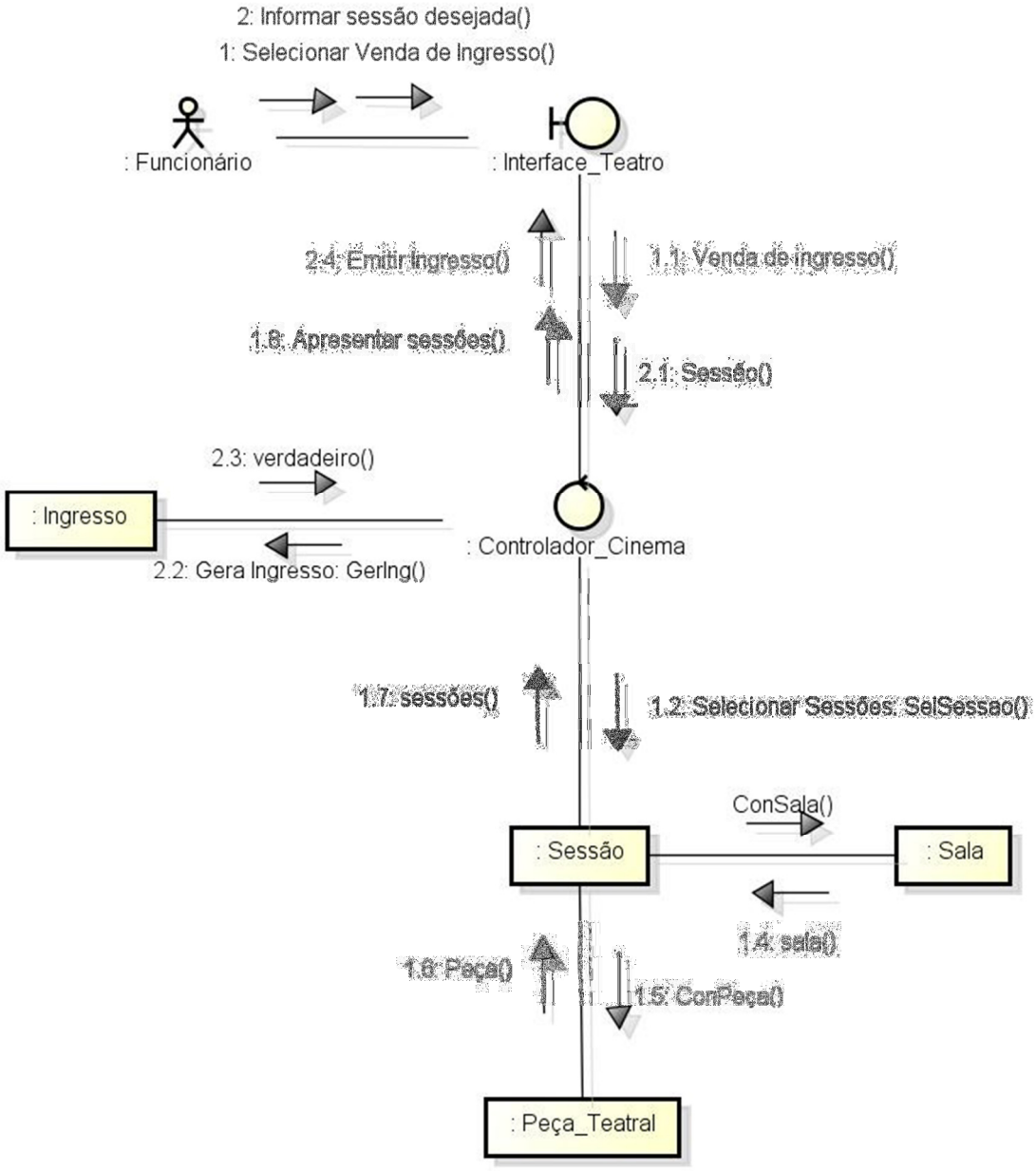
Fonte: Próprio autor

4.2.4 Diagrama de comunicação

O diagrama de comunicação era conhecido até a versão 1.5 da UML, como diagrama de colaboração, seu nome foi modificado a partir da versão 2.0.

Este diagrama está fortemente associado ao diagrama de sequência, pois normalmente as informações contidas no diagrama de sequência também serão mostradas no diagrama de comunicação, a diferença está em seu enfoque, o diagrama de sequência não se preocupa com a temporalidade do processo, e tem seu foco na maneira como os elementos estarão vinculados e nas mensagens que serão trocadas entre si. Apesar de terem basicamente os mesmos atores e objetos, o diagrama de comunicação não possui linhas de vida (GUEDES, 2009, p. 239), como demonstrado na Figura 12.

Figura 12 - Exemplo de diagrama de comunicação



Fonte: Próprio autor

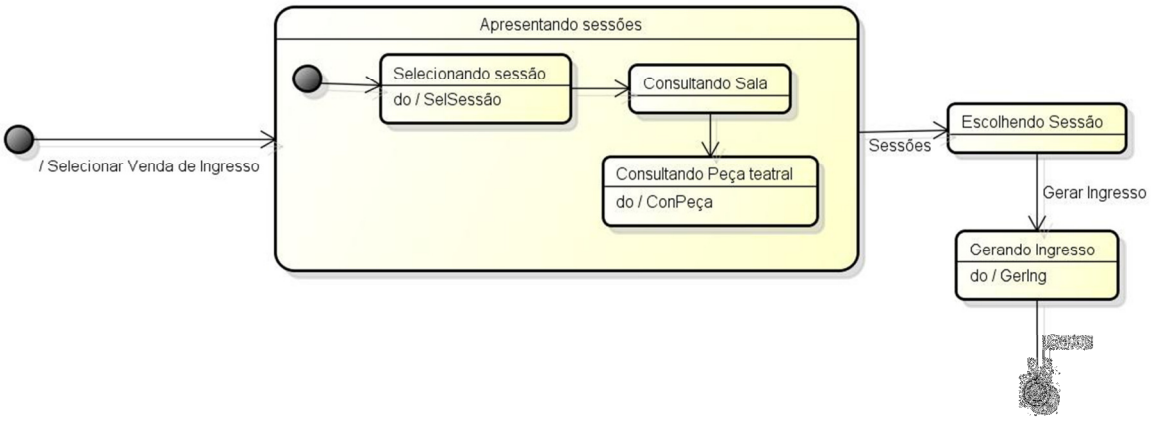
4.2.5 Diagrama de máquina de estados

"Uma máquina de estados é um comportamento que especifica as sequências de estados pelas quais um objeto passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos" (BOOCH; RUMBAUGH; JACOBSON, 2005, p. 299).

Segundo Guedes (2009, p. 250) um estado representa a situação de um elemento, ou um objeto, durante certo período de sua vida do qual participa de um processo (podendo passar por muitos estados dentro de um mesmo processo), conforme a Figura 13. Segundo ele, um estado pode demostrar:

a. A espera para o acontecimento de um evento.
b. A resposta a um estímulo.
c. A realização de alguma atividade.
d. A satisfação de alguma condição.

Figura 13 - Exemplo de diagrama de máquina de estados



Fonte: Próprio autor

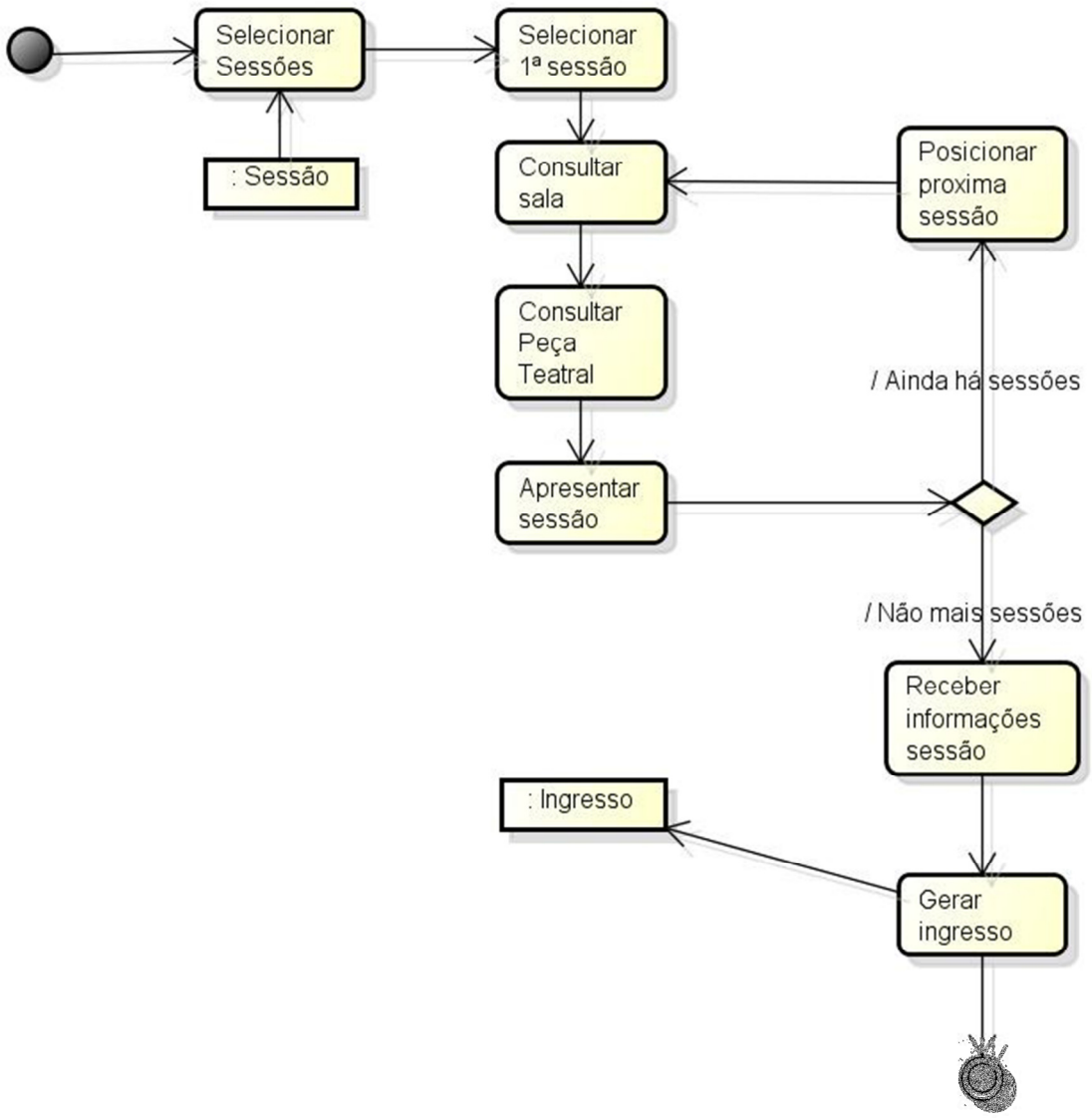
4.2.6 Diagrama de atividades

Um diagrama de atividade é basicamente um gráfico de fluxo, pois demonstra um fluxo de atividade para outra, porém um diagrama de atividade também demonstra concorrência e suas ramificações de controle (BOOCH; RUMBAUGH; JACOBSON, 2005, p. 265).

Uma atividade é formada pelo conjunto de ações para que uma atividade seja concluída, dessa maneira, um diagrama de atividade pode modelar mais de uma atividade (GUEDES, 2009, p. 285).

Booch, Rumbaugh e Jacobson (2005, p.269) explana que os diagramas de atividades são importantes não somente para a modelagem dos aspectos dinâmicos do sistema, mas também para que seja possível construir sistemas executáveis fazendo o uso de engenharia de produção reversa, de acordo com a Figura 14.

Figura 14 - Exemplo de diagrama de atividades



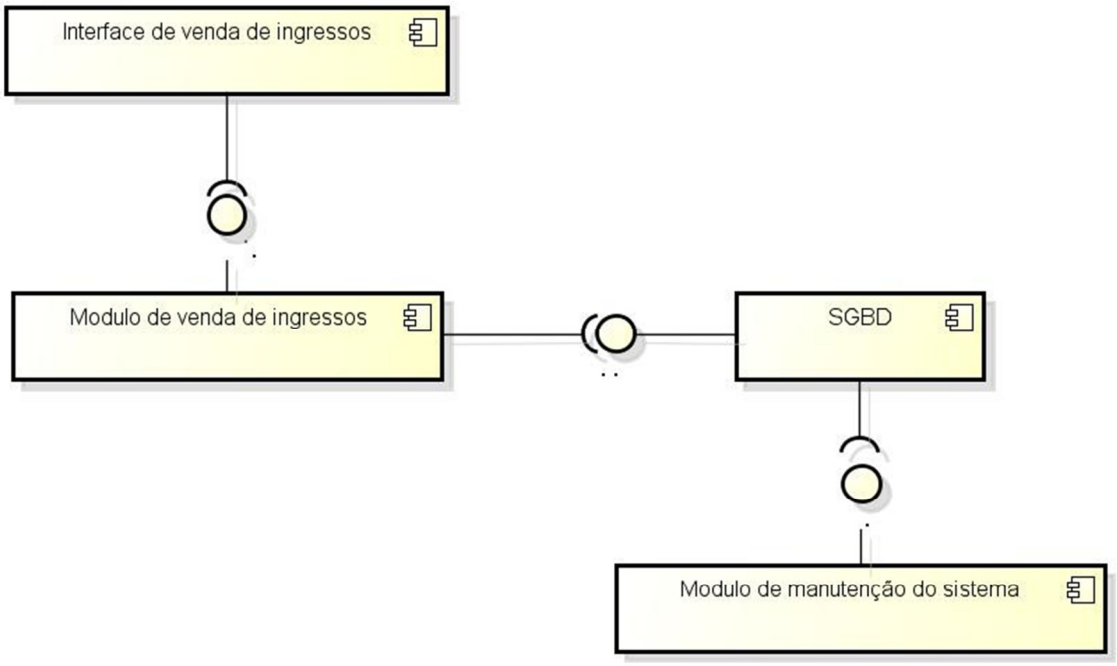
Fonte: Próprio autor

4.2.7 Diagrama de componentes

Um componente é a parte lógica de um sistema ou subsistema, podendo ter uma ou mais interfaces “fornecidas e requeridas (potencialmente exposta via portas), e seus interiores são transparentes e inacessíveis por outro meio que não seja fornecido por suas interfaces” (GUEDES, 2009, p. 329).

Através dos componentes, como se pode ver na Figura 15, é possível que encapsular partes do sistema para reproduzir as dependências e torna-las expostas para que caso seja necessário uma substituição o sistema terá mais flexibilidade, explicam Booch, Rumbaugh e Jacobson (2005, p.196).

Figura 15 - Exemplo de diagrama de componentes



Fonte: Próprio autor

5 CONSIDERAÇÕES FINAIS

O objetivo geral deste trabalho foi fornecer uma documentação adequada e apresentar a modelagem de sistema em UML para uma empresa fictícia de pequeno porte, mostrando os diagramas essenciais para o desenvolvimento de um software para empresa. Dessa forma, foi possível alcançar os objetivos estabelecidos, através do estudo e uso da Engenharia de Software e UML.

A versão 2 da UML possui diversos diagramas, a fim de que se possa oferecer tanto ao cliente quanto ao desenvolvedor, diversas visões do mesmo sistema (ou parte dele, como por exemplo, uma funcionalidade específica), facilitando assim que o cliente confirme que o que está documentado de fato está de acordo com a sua necessidade, e também que os desenvolvedores consigam atingir as expectativas.

Através do estudo bibliográfico sobre Engenharia de Software e UML, foi possível ampliar a compreensão sobre o tema, e assim, desenvolver-se o estudo de caso para o teatro.

Os objetivos gerais desse trabalho foram atingidos, onde a tecnologia UML foi utilizada na modelagem do software. Baseando se nos estudos de caso feitos por Guedes, a autora não utilizou todos os diagramas contidos na UML, por não se aplicarem a situação desse sistema de complexidade menor, dessa maneira a repetição das informações contidas nos diagramas pode ser evitada, agilizando o início do desenvolvimento e também o custo com essa documentação.

Ao longo deste trabalho foram identificadas novas possibilidades de trabalhos futuros que não foram realizadas no mesmo, pois tornariam o mesmo muito extenso. A primeira delas é o desenvolvimento dos diagramas citados neste trabalho para todas as funcionalidades do sistema, utilizando-se como base os diagramas realizados no estudo de caso, a saber, a funcionalidade de venda de ingressos. A implantação deste sistema também pode ser objeto de estudo futuro, para se

observar a prática do desenvolvimento fazendo o uso dos diagramas utilizados nessa monografia, pois ainda há no mercado atual diversas empresas de pequeno porte sem documentação adequada de seus softwares devido ao custo e trabalho, dificultando assim, que ele seja adaptado as necessidades do cliente de tempos em tempos.

REFERÊNCIAS

REZENDE, DENIS ALCIDES. **Engenharia de Software e Sistema de Informação**. 3. ed. Rio de Janeiro: Editora BRADSPORT, 2005.

BRASIL. **Lei complementar n. 123, de 14 de dezembro de 2006**. Institui o estatuto nacional da microempresa e da empresa de pequeno porte. Brasília, DF. Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/lcp/lcp123.htm>. Acesso em: 03 mar. 2014

BNDES. **Porte de empresa.** BNDES, O Banco Nacional do Desenvolvimento. Disponível em: <http://www.bndes.gov.br/SiteBNDES/bndes/bndes_pt/Institucional/Apoio_Financeiro/porte.html>. Acesso em: 03 mar. 2014

SEBRAE. **Critério de classificação em empresas: EI – ME – EPP.** Disponível em: http://www.sebrae-sc.com.br/leis/default.asp?vcdtexto=4154. Acesso em: 03 mar. 2014

PRESSMAN, Roger S.. **Engenharia de Software.** 1. ed. São Paulo: Editora Makron Book, 1995.

PRESSMAN, Roger S.. **Engenharia de software: uma abordagem profissional.** 7. ed. Porto Alegre: AMGH, 2011.

SOMMERVILLE, Ian. **Engenharia de Software.** Editora Pearson. Ano: 2003

LAUSON, K C.; LAUDON, J P. **Sistemas de informação gerenciais.** 9. Ed. São Paulo: Pearson Prentice Hall, 2010.

O'BRIEN, J A. **Sistemas de informação e as decisões gerenciais na era da internet.** 2.ed. São Paulo: Saraiva, 2004.

STAIR, R .M.; REYNOLDS, G W. **Princípios de sistemas de informação.** 9.ed. São Paulo: Cengage Learning, 2011.

GUEDES, GILLEANES , T. A. **UML 2: uma abordagem prática.** São Paulo: Novatec Editora, 2009.

BOOCH, GRADY; RUMBAUGH, JAMES; JACOBSON, IVAR. **UML: guia do usuário.** 2. ed. Rio de Janeiro: Elsevier, 2005.