VISAR3D-DYNAMIC: UMA ABORDAGEM PARA APOIAR A COMPREENSÃO DO COMPORTAMENTO DINÂMICO DE SOFTWARE POR MEIO DE REALIDADE VIRTUAL

Filipe Arantes Fernandes

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadoras: Cláudia Maria Lima Werner
Claudia Susie Camargo Rodrigues

Rio de Janeiro
Abril de 2017

VISAR3D-DYNAMIC: UMA ABORDAGEM PARA APOIAR A COMPREENSÃO DO COMPORTAMENTO DINÂMICO DE SOFTWARE POR MEIO DE REALIDADE VIRTUAL

Filipe Arantes Fernandes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____
Profª. Cláudia Maria Lima Werner, D.Sc.

_____
Profª. Claudia Susie Camargo Rodrigues, D.Sc.

_____
Prof. Geraldo Bonorino Xexéo, D.Sc.

_____
Profª. Debora Christina Muchaluat Saade, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 2017

Fernandes, Filipe Arantes

VisAr3d-Dynamic: Uma Abordagem para Apoiar a Compreensão do Comportamento Dinâmico de Software por meio de Realidade Virtual/Filipe Arantes Fernandes. – Rio de Janeiro: UFRJ/COPPE, 2017.

XVII, 120 p.: il.; 29,7 cm.

Orientadoras: Cláudia Maria Lima Werner

Claudia Susie Camargo Rodrigues

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2017.

Referências Bibliográficas: p. 96-106.

1. Análise Dinâmica de Software. 2. Compreensão de Programas. 3. Realidade Virtual. 4. Visualização de Software. I. Werner, Cláudia Maria Lima *et al.*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

A meus pais José Carlos (*in memorian*) e Jane e meus avós.

AGRADECIMENTOS

"...todas as coisas cooperam para o bem daqueles que amam a Deus..." Rm 8.28. Agradeço a Deus, pelo cuidado e amor que demostrou durante minha jornada até aqui. Foram muitas horas viajando pelas estradas sem que haja qualquer tipo de acidente e proteção em meio aos riscos da violência da cidade do Rio de Janeiro.

Apesar de não estar mais entre nós, gostaria de registrar a minha imensa gratidão ao meu pai, José Carlos Fernandes. Infelizmente, ele não está acompanhando minha ascensão tanto profissional quanto pessoal, mas mesmo assim, o agradeço pelo incentivo e amor que demonstrou enquanto estava presente. Foi motivo de força quando vinham as dificuldades.

À minha mãe, Jane, pelos ensinamentos que me passou e continuam me transmitindo mesmo depois de "velho", pelas palavras de incentivo, pelo carinho e, principalmente, pela calma e compreensão durante a escrita da dissertação. Foram momentos de tensão, que, com sabedoria soube lidar comigo. Obrigado, mãe!

Aos meus parentes em geral, que de certa forma me apoiaram nesta caminhada. Especialmente, quero registrar o carinho que tiveram comigo, mas que infelizmente eu os perdi durante a realização deste trabalho, meu tio Nício e minha vovó Lulu.

À minha orientadora, Cláudia Werner, pela colaboração, ensinamentos, atenção, pelas oportunidades oferecidas e muita paciência para com este mero aluno. Sem sua orientação, seria muito difícil obter os resultados alcançados. Espero que, de certa forma, tenha contribuído face a confiança em mim depositada.

À minha coorientadora, Claudia Susie, que iniciou este desafio na COPPE/UFRJ de "misturar" Realidade Virtual e Engenharia de Software. Obrigado por ser visionária e acreditar neste novo campo de atuação. Além disso, agradeço pelas várias conversas e conselhos sobre a vida e, é claro, sobre a pesquisa também.

Aos meus amigos de longa data Jean, Patrick e Robson Júnior que, apesar de encontrarmos poucas vezes devido a distância, me proporcionaram momentos agradáveis e conversas muito descontraídas.

Aos diversos colegas pelo apoio e momentos de alegria que me proporcionaram.

Aos meus colegas do LENS/PESC, Sérgio e Willian, por terem me ajudado na condução de alguns procedimentos burocráticos. À Thaiana pela amizade e apoio durante o mestrado.

Aos amigos do grupo de reutilização e do Lab3D da COPPE/UFRJ, pela convivência e amizade.

Aos funcionários do PESC/COPPE que, com muita dedicação e competência, apoiaram indiretamente o desenvolvimento da pesquisa preparando cafés, mantendo nossos laboratórios limpos e apoiando em nossos processos burocráticos.

Aos colegas da COPPE/UFRJ e IComp/UFAM por terem aceitado o convite de participar do estudo de observação desta dissertação de mestrado.

À CAPES, pelo apoio financeiro durante o mestrado.

A todos aqueles que passaram em minha vida e contribuíram de alguma forma.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

VISAR3D-DYNAMIC: UMA ABORDAGEM PARA APOIAR A COMPREENSÃO DO COMPORTAMENTO DINÂMICO DE SOFTWARE POR MEIO DE REALIDADE VIRTUAL

Filipe Arantes Fernandes

Abril/2017

Orientadoras: Cláudia Maria Lima Werner
Claudia Susie Camargo Rodrigues

Programa: Engenharia de Sistemas e Computação

Sistemas de software são desenvolvidos e mantidos por um período considerável de tempo por uma equipe de desenvolvedores com alta rotatividade. As alterações feitas por desenvolvedores que não entendem o *design* original podem causar a degradação do software. Após essas mudanças, necessita-se conhecer tanto o *design* original quanto as novas regras implementadas, a fim de entender o produto. Depois de várias alterações, tanto os engenheiros que projetaram o software quanto desenvolvedores que não estão familiarizados com o sistema enfrentarão dificuldades de entender o software, pois o que é executado é incompatível com sua documentação.

Uma vez que a compreensão é uma atividade crítica e desafiadora, o desenvolvimento de técnicas e ferramentas que apoiem esta atividade pode influenciar na eficiência geral do desenvolvimento de software. O objetivo principal desta dissertação é propor uma abordagem, denominada VisAr3D-Dynamic, a qual visa apoiar a compreensão do comportamento dinâmico de sistemas de software de larga escala por meio de Realidade Virtual, provendo recursos interativos durante a análise dinâmica de software. Esta abordagem teve seu desempenho avaliado por um estudo de observação, cujos resultados fornecem indicadores positivos em relação à compreensão do comportamento dinâmico de sistemas de software.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

VISAR3D-DYNAMIC: AN APPROACH FOR SUPPORTING THE COMPREHENSION OF THE DYNAMIC BEHAVIOR OF SOFTWARE THROUGH VIRTUAL REALITY

Filipe Arantes Fernandes

April/2017

Advisors: Cláudia Maria Lima Werner
Claudia Susie Camargo Rodrigues

Department: Computer and Systems Engineering

Software systems are developed and maintained for a considerable period of time by a team of highly rotating developers. Changes made by developers who do not understand the original design can cause software degradation. After these changes, it is necessary to know both the original design and the new rules implemented in order to understand the product. After several changes, both engineers who designed the software and developers who are unfamiliar with the system will have difficulty in understanding the software because what is executed is incompatible with its documentation.

Since understanding is a critical and challenging activity, the development of techniques and tools that support this activity can influence the overall efficiency of software development. The main objective of this dissertation is to propose an approach, named VisAr3D-Dynamic, which aims to support the understanding of the dynamic behavior of large-scale software systems through Virtual Reality, providing interactive resources during software dynamic analysis. This approach had its performance evaluated by an observation study, whose results show positive indicators regarding the understanding of the dynamic behavior of software systems.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

CAPÍTULO 1 - INTRODUÇÃO

1.1. Motivação

A manutenção de software é considerada como a última fase do ciclo de vida do desenvolvimento de software. Após a liberação e entrega do produto, os mantenedores garantem a atualização do software em consonância aos pedidos de alteração dos *stakeholders*, bem como às mudanças ocorridas no ambiente. No entanto, à medida que o sistema evolui ao longo do tempo, os conceitos implementados tendem a divergir dos documentos iniciais do desenvolvimento. Em decorrência disto, grande parte dos projetos de desenvolvimento, apresentarão documentação não confiável e desatualizada sobre a estrutura e comportamento do sistema em um nível de abstração que seja compreensível pelos engenheiros de software (LEHMAN, 1980; LEHMAN, 1996; LIENTZ *et al.*, 1978; LIENTZ & SWANSON, 1980).

Frequentemente, os sistemas de software são desenvolvidos e mantidos por um período considerável de tempo por uma equipe de desenvolvedores com alta rotatividade. As alterações feitas por desenvolvedores que não entendem o *design* original podem causar a degradação do software, ou seja, inconsistências entre os conceitos projetados e implementados (MENS *et al.*, 2005). Após essas mudanças, necessita-se conhecer tanto o *design* original quanto as novas regras implementadas, a fim de entender o produto. Depois de várias alterações, tanto os engenheiros que projetaram o software quanto desenvolvedores que não estão familiarizados com o sistema enfrentarão dificuldades de entender o software, pois o que é executado é incompatível com sua documentação. Deste modo, torna-se caro manter atualizados sistemas que são repetidamente modificados (PARNAS, 1994).

1.2. Problema

A fim de manter adequadamente um sistema de software, todos os mantenedores devem compreender suficientemente o funcionamento do sistema. Caso esse conhecimento não esteja prontamente disponível (*e.g.*, documentação), a equipe é confrontada com a difícil tarefa de obter uma compreensão do funcionamento interno do sistema, ou seja, entender quais trechos do código-fonte são executados (LEHMAN & BELADY, 1985). Segundo BIGGERSTAFF *et al.* (1993), este processo é conhecido como compreensão de programas.

Tipicamente, a compreensão de programas consiste no estudo de artefatos, tais como código-fonte e documentação. No entanto, lidar com código-fonte envolve um mapeamento mental entre o código do sistema e seu comportamento. Nos casos de sistemas que apresentam muitas anomalias inerentes ao desenvolvimento, cujo o código-fonte possui centenas ou até milhares de linhas (BENNETT & RAJLICH, 2000), são difíceis de serem interpretados diretamente, pois resultam em uma sobrecarga cognitiva por parte do mantenedor.

Além disso, a documentação do software pode estar incompleta, desatualizada ou, em casos extremos, inexistir (VON MAYRHAUSER & VANS, 1995; LETOVSKY, 1987). Como consequência, a compreensão de programas é uma atividade bastante demorada e custosa. A literatura técnica relata que até 60% do esforço de engenharia de software é concentrado na compreensão de sistemas de software (FJELDSTAD & HAMLEN, 1979; CORBI, 1989; PIGOSKI, 1997). Neste sentido, manter adequadamente sistemas complexos emerge como um grande desafio, pois demanda a utilização de técnicas e ferramentas a fim de apoiarem a compreensão, reduzindo a sobrecarga cognitiva e aumentando a eficiência geral do desenvolvimento de software.

Nesta dissertação de mestrado, um sistema de larga escala ou sistema complexo, é definido como qualquer sistema de software composto por um número elevado de elementos com muitas interações (SOMMERVILLE *et al.*, 2012).

1.3. Questão de Pesquisa

Uma vez que a compreensão é uma atividade crítica e desafiadora, o desenvolvimento de técnicas e ferramentas que apoiem esta atividade pode influenciar na eficiência geral do desenvolvimento de software. A literatura técnica oferece diversos métodos: análise de rotas de execução, reconstrução da arquitetura, localização de características, dentre outras. A maioria das abordagens podem ser divididas em análises estáticas e dinâmicas (e suas combinações) (CORNELISSEN *et al.*, 2009).

As abordagens estáticas estão relacionadas com a análise de artefatos estruturais, tal como o código-fonte. Contudo, este tipo de abordagem desconsidera o comportamento dinâmico do software, destacando-se como uma de suas principais desvantagens. Por exemplo, em sistemas orientados a objetos, ocorrências de ligação tardia e polimorfismo são difíceis de serem compreendidos se as informações de

execução estiverem ausentes (MATHIASSEN *et al.*, 2000; BREIVOLD & LARSSON, 2007).

Por outro lado, a análise dinâmica tem como objetivo examinar as propriedades de execução de um sistema de software (BALL, 1999). Os dados resultantes podem ser usados para diversos fins, tanto para engenharia reversa quanto para depuração, muitas vezes sob a forma de rastros de execução. Dentre suas vantagens, destacam-se a precisão em relação ao comportamento real do sistema de software (BALL, 1999) e na utilização de cenários de execução como estratégia orientada a objetivos, a qual somente as partes de interesse do sistema são analisadas (KOENEMANN & ROBERTSON, 1991; ZAIDMAN, 2006).

Apesar de suas vantagens, esta técnica apresenta algumas limitações, tais como a incompletude, pois cada cenário de execução consegue capturar apenas uma fração do domínio (BALL, 1999); a dificuldade de determinar quais cenários executar para desencadear os elementos de interesse do programa (BALL, 1999); e a escalabilidade, devido ao grande volume de dados que podem ser produzidos, afetando o desempenho, o armazenamento e, sobretudo, a sobrecarga cognitiva (ZAIDMAN, 2006).

Para lidar com questões de escalabilidade e sobrecarga cognitiva na análise dinâmica, muitas abordagens foram propostas focando somente na representatividade das informações, ou seja, no estudo de metáforas visuais mais eficazes. No entanto, devido ao grande volume de dados da análise dinâmica, o desafio de melhor representar os dados da execução do software de forma escalável e compreensível ainda persiste. Neste sentido, com base nas limitações e nos desafios apresentados, esta dissertação de mestrado conduziu o trabalho a partir da seguinte questão de pesquisa:

Como apoiar a compreensão de um grande volume de dados gerados a partir da análise dinâmica de software?

O uso da representação gráfica dos dados pode ajudar significativamente na análise e na compreensão de sistemas complexos (DIEHL, 2007). A visualização de software consiste em criar imagens de software por meio de objetos visuais. Esses objetos visuais podem representar, por exemplo, sistemas, componentes ou o comportamento em tempo de execução. Representações gráficas eficazes são capazes de fornecer melhores *insights* do que elementos textuais (CHEN, 2006) e podem aumentar a compreensão e redução dos custos de desenvolvimento (MILI & STEINER, 2002).

As ferramentas de visualização precisam levar em consideração os objetivos que desejam alcançar. MALETIC *et al.* (2002) dividem essas características em cinco dimensões de uma visualização: tarefa, audiência, alvo, representação e meio. Essas dimensões englobam as classificações propostas por taxonomias de visualizações muito referenciadas (PRICE *et al.*, 1993; ROMAN & COX, 1993), e correspondem ao *porquê*, *quem*, *o que*, *como* e o *meio* da visualização, respectivamente. A tarefa diz respeito do *porquê* da necessidade da visualização; a audiência está relacionada a *quem* utiliza a visualização; o alvo define *o que* é visualizado; a representação trata sobre *como* os dados serão exibidos e, por fim, o meio é *onde* a visualização é observada pelo usuário.

Na literatura técnica, grande parte das ferramentas de visualização de software não exploram locais alternativos para a exibição das visualizações, tão pouco alternativas de interação. Geralmente, sistemas de visualização utilizam monitores com resolução mediana, não explorando outras técnicas de projeção e interação, tais como monitores com alta resolução, *display wall,* bem como interfaces avançadas de usuário por meio da Realidade Virtual (RV).

1.4. Hipótese e Objetivo da Pesquisa

A RV é uma interface avançada para aplicações computacionais, onde o usuário pode navegar e interagir, em tempo real, em um ambiente tridimensional gerado por computador, usando dispositivos multissensoriais (KIRNER & TORI, 2004). Em ambientes virtuais, o usuário tem a impressão de estar atuando dentro destes ambientes em tempo real. Em virtude destas características, a RV tem sido utilizada por diversas áreas com aplicabilidades distintas, inclusive na visualização de dados complexos (VAN DAM *et al.*, 2000). Usuários podem facilmente explorar e compreender estruturas ou representações gráficas de um grande volume de dados por meio de uma experiência imersiva e interativa.

A problemática que esta dissertação de mestrado busca solucionar é a redução da sobrecarga cognitiva durante a compreensão de um grande volume de dados gerados a partir do comportamento dinâmico de software. Considerando os desafios da análise dinâmica e o potencial da RV na visualização de dados complexos, foi elaborada a seguinte hipótese de pesquisa:

A Realidade Virtual pode apoiar na compreensão do comportamento dinâmico de software em relação às visualizações tradicionais.

O objetivo principal desta dissertação é propor uma abordagem, denominada VisAr3D-Dynamic, a qual visa apoiar a compreensão do comportamento dinâmico de sistemas de software de larga escala por meio de Realidade Virtual, provendo recursos interativos durante a análise dinâmica de software. De forma que a hipótese de pesquisa seja testada, será desenvolvido uma ferramenta em RV que possibilite a exploração de informações sobre o comportamento dinâmico de sistemas complexos e, posteriormente, a condução de um estudo para verificar o ganho da RV à compreensão de programas através da análise dinâmica de software.

1.5. Estrutura da Dissertação

Esta dissertação está organizada em seis capítulos. O presente capítulo apresentou a motivação para o desenvolvimento deste trabalho, bem como o problema, a questão, hipótese e o objetivo da pesquisa.

O Capítulo 2 aborda a Realidade Virtual como principal técnica inovadora em análise dinâmica de software. São apresentados os principais elementos de um sistema de RV, além de uma breve discussão sobre visualização em 3D e visualização em RV.

O Capítulo 3 trata sobre a visualização de software, mais especificamente na visualização do comportamento dinâmico do software, bem como a apresentação e discussão de alguns trabalhos relacionados, agrupados quanto ao tipo de representação.

No Capítulo 4, propõe-se a abordagem para apoiar a compreensão do comportamento dinâmico de software por meio de RV. O contexto da pesquisa, a visão geral e os recursos da abordagem também são apresentados neste capítulo.

O Capítulo 5 apresenta os detalhes de implementação da abordagem proposta por esse trabalho por meio de um protótipo capaz de apoiar o comportamento dinâmico de sistemas de software sob determinado cenário de execução.

O Capítulo 6 discute o método e apresenta os resultados de uma avaliação inicial da abordagem proposta, com o objetivo de evidenciar os ganhos obtidos pela adoção dessa estratégia, como apoio na compreensão do comportamento dinâmico de sistemas de software orientados a objetos.

Por fim, o Capítulo 7 contém as considerações finais deste trabalho, bem como as contribuições da dissertação, algumas limitações identificadas e as perspectivas de trabalhos futuros.

CAPÍTULO 2 - REALIDADE VIRTUAL

2.1. Introdução

Na década de 1950, coube a um cineasta a concepção do primeiro dispositivo que propiciava a imersão dos sentidos dos participantes em um mundo virtual tridimensional; a um engenheiro, no final da década de 1960, a construção do primeiro capacete de Realidade Virtual (RV) e a um artista e cientista da computação, no final da década de 1980, a proposta do termo ao qual conhecemos como RV (TORI & KIRNER, 2006).

Na literatura técnica, pesquisadores definem RV à luz de suas perspectivas e disciplinas de interesses. Por exemplo, PIMENTEL & TEIXEIRA (1993) definiram RV como uma experiência imersiva e interativa gerada por um computador. Para BROOKS (1999), RV são sistemas computacionais que proporcionam imersão ao usuário a um mundo virtual. ZAHO (2002) define como um sistema baseado em computador, ao qual consiste em um ambiente virtual, em ambiente físico, bem com software e hardware, que permite a interação entre um ser humano e um computador. SHERMAN E CRAIG (2002) consideram RV como um meio composto por simulações interativas baseadas por computador. Através da interface, a posição e ações dos participantes são detectadas, a fim de substituir ou aumentar o *feedback* de um ou mais sentidos. DIONISO & GILBERT (2013) conceituam RV como simulações geradas por computador de ambientes e objetos tridimensionais com aparência real e interação física do usuário.

De fato, existem muitas definições de RV envolvendo aspectos gerais e conceitos tecnológicos. No entanto, uma definição que engloba estas características é apresentada por TORI & KIRNER (2006), que consideram RV como uma interface avançada para aplicações computacionais, que permite ao usuário navegação e interação em tempo real, em um ambiente tridimensional, podendo fazer uso de dispositivos multissensoriais, para atuação ou *feedback*.

Paralelo a isso, SHERMAN & CRAIG (2002) apresentaram quatro elementos-chave de um sistema de RV: ambiente virtual, imersão, *feedback* sensorial e interatividade. Adicionalmente, BROOKS (1999) considera os participantes também

como um elemento-chave para sistemas de RV. Portanto, o restante desta seção irá apresentar um panorama sobre RV por meio destes elementos-chave.

2.2. Elementos-Chave de um Sistema de Realidade Virtual

2.2.1. Ambiente Virtual

O Ambiente Virtual (AV) – também conhecido como mundo virtual – é um espaço tridimensional gerado por computador, onde usuários interagem entre si (por meio de avatares) ou com outros objetos virtuais (BIOCCA & LEVY, 2003). SHERMAN & CRAIG (2002) definem como o conteúdo de algum meio. Os objetos virtuais podem manifestar certos atributos, tais como geometria, cores, texturas, iluminação, características dinâmicas, restrições físicas e atributos acústicos (TORI & KIRNER, 2006).

Em alguns AVs podem ser utilizadas precisões geométricas, tais como cores, texturas e iluminação, as quais são elementos importantes para simular o mundo real. No entanto, certos AVs não referenciam ao mundo real, constituindo-se apenas de modelos abstratos. Independentemente do tipo de ambiente, as cores, texturas e iluminação são elementos importantes para uma boa visualização e imersão. Há situações em que o AV é utilizado para avaliar apenas alguma simulação comportamental, na qual a precisão do comportamento é mais importante que a fidelidade visual. É o caso de reações químicas, que podem usar representações simples das moléculas baseadas em esferas coloridas, por exemplo (TORI & KIRNER, 2006).

2.2.2. Imersão

A psicologia define imersão como um estado adquirido pelo participante, quando há percepção de estar envolvido por um espaço virtual durante a realização de uma determinada atividade (MUHANNA, 2015). A imersão é comumente utilizada para descrever a sensação de “estar dentro” de ambientes virtuais. No entanto, diversos autores definem e classificam a imersão em diferentes perspectivas.

SHERMAN & CRAIG (2002) definem imersão como a sensação de estar em um ambiente; podendo ser puramente um estado mental ou acompanhado por algum meio físico. A imersão mental pode ser adquirida durante a leitura de um livro, transportando o leitor a um novo mundo inexistente, transmitindo a sensação de fazer parte dele, ou a empatia com personagens e outros elementos, os quais fazem esquecer do mundo real e arredores. Este cenário pode levar a uma experiência imersiva, devido ao fato de estar

engajado em uma atividade. Ao contrário da imersão mental, a imersão física é experimentada por meio de estímulos sensoriais apoiados por recursos tecnológicos. Por exemplo, no treinamento de pilotos de aeronaves, podem ser utilizados cabines reais para simulação de voos. Mediante à interação com os elementos da cabine, os pilotos recebem a resposta por meio da visão ou tato (simulação de turbulência) proporcionando uma experiência real no comando da aeronave, como mostra a Figura 2.1. Deste modo, sistemas de RV proporcionam imersão mediante a interação e resposta por meio de dispositivos tecnológicos.

Figura 2.1 – Participantes interagindo com um simulador de voo (MUHANNA, 2015)

NAKATSU & TOSA (2000) introduziram os termos de imersão passiva e imersão ativa. A ausência ou a existência de interação é o elemento-chave que distingue os dois tipos de imersão. A imersão ativa inclui a interação com objetos, enquanto que na imersão passiva os usuários só recebem informações sem interação. Observando um filme, por exemplo, pode ser considerado um exemplo de imersão passiva. Portanto, uma experiência de RV deve envolver uma imersão ativa alcançada através da implementação de diferentes abordagens de interação.

TORI & KIRNER (2006) classificam sistemas de RV, em função do senso de presença, em imersivo e não-imersivo. A RV é imersiva quando o usuário é transportado predominantemente para o domínio da aplicação, através de dispositivos multissensoriais, que capturam seus movimentos e comportamento, provocando uma sensação de presença dentro do mundo virtual. Quando o usuário é transportado parcialmente ao mundo virtual, mas continua a sentir-se predominantemente no mundo real, o sistema de RV é considerado não-imersivo. A visualização de AVs com monitores é um exemplo, pois não isola completamente o participante do mundo

externo, ao passo que a utilização de um capacete de RV é considerada pelos autores como imersiva.

2.2.3. Feedback

Outra característica importante em sistemas de RV é o *feedback* sensorial ou *feedback*. Este elemento diz respeito a capacidade de os participantes observarem os resultados de suas atividades (SHERMAN & CRAIG, 2003). Utilizando um capacete de RV, por exemplo, ao qual o isola totalmente da realidade, o AV deve atualizar a imagem exibida de acordo com a movimentação da cabeça. Em outras palavras, se o participante olhar para a direita, o display deve mostrar o que existe no lado direito do AV, e assim por diante. Isto deve ser feito em um período de tempo realista e sem atrasos.

Além do visual, o *feedback* tátil também é importante, como é o caso do simulador para prática de sutura. WEBSTER *et al.* (2001) desenvolveu um sistema para ensinar o usuário a realizar uma sutura utilizando uma agulha especial. Durante a simulação, o usuário pode perceber as reações elásticas da pele ao puxar a linha, bem como observar o melhor local para inserção dos pontos. O simulador oferece um modelo monoscópico da pele adicionado de propriedades físico-elásticas, de forma a oferecer retorno tátil e de força ao usuário durante a manipulação de um dispositivo háptico[1].

Normalmente, os atrasos admissíveis para que o ser humano tenha a sensação de interação em tempo real estão em torno de 100 milissegundos, tanto para a visão, quanto para as reações de tato, força e audição. Isto impõe um compromisso do sistema (processadores, software, dispositivos, complexidade do ambiente virtual, tipo de interação, dentre outros) em funcionar com taxas mínimas de 10 quadros por segundo na renderização das imagens (sendo desejado algo em torno de 20 quadros por segundo para suportar melhor as cenas animadas) e de 100 milissegundos de atraso nas reações aos comandos do usuário. Assim, a complexidade do mundo virtual, os dispositivos usados, o software e a configuração do sistema devem ser ajustados para funcionar com as taxas mínimas de renderização e reação (TORI & KIRNER, 2006).

[1] Dispositivos de reação tátil que procuram estimular sensações como o tato, tensão muscular e temperatura.

2.2.4. Interatividade

Em uma experiência de RV, a interatividade dá aos participantes a capacidade de interagir e modificar o AV. A interatividade é alcançada por meio de dispositivos multissensoriais que permitem aos participantes interagirem dinamicamente com objetos virtuais através da navegação, manipulação direta ou outros estilos de interação (MUHANNA, 2015). BOWMAN *et al.* (2004) identificaram quatro técnicas de interação com AV complexos:

- Navegação: refere-se à movimentação do usuário dentro do AV. Esta técnica envolve uma "viagem", que consiste na movimentação mecânica no ambiente e na definição de um trajeto. A viagem é usada para exploração e busca envolvendo posicionamento, objetividade, velocidade, aceleração, dentre outros recursos interativos. A definição do trajeto é um processo de tomada de decisão, que permite o estabelecimento de um caminho a ser seguido. Ele depende do conhecimento e do comportamento espacial do usuário e de elementos de ajuda, tais como mapas, objetos de referência, além de elementos de áudio;

- Seleção: consiste na escolha de um objeto virtual a ser manipulado. Envolve três passos: indicação do objeto, confirmação e realimentação. A indicação pode ocorrer por oclusão, toque no objeto, apontamento ou de maneira indireta. O sistema deve mostrar a seleção, usando elementos visuais, auditivos ou hápticos, tais como mudança de cor, alertas visuais ou sonoros etc. Para que a seleção tenha efeito, deve ser confirmada com eventos disparados por dispositivos, como por exemplo, *mouse*, teclado, comando de voz, gestos ou outras ações. E, por fim, deve existir uma realimentação informando que a ação foi realizada;

- Manipulação: consiste na alteração de um objeto selecionado para atender diversos propósitos, tais como a alteração de posição, por meio de translação ou rotação, ou de suas características, envolvendo escala, cor, transparência e textura. O objeto selecionado pode ser também apagado, copiado, duplicado ou alterado por outras ações;

- Controle: consiste na emissão de comandos do usuário a serem executados pelo sistema. Os comandos podem ser emitidos, por meio de menus gráficos, comandos de voz, comandos gestuais ou por meio de dispositivos de comandos específicos.

A interação e o *feedback* são elementos que devem funcionar em plena harmonia, pois são vitais para sistemas de RV. Ambos devem ser realizados em tempo real, caso contrário, resultará em uma experiência negativa no AV, o qual é objeto de exploração e interação (PREDDY & NANCE, 2002).

2.2.4.1. Dispositivos

Dispositivos convencionais, tais como teclado e *mouse*, podem ser utilizados pelos participantes como forma de interação com AVs. No entanto, dispositivos não convencionais possuem a característica de facilitar a interação com objetos virtuais e tridimensionais, bem como potencializar a sensação de imersão utilizando até os cinco sentidos humanos simultaneamente. Estas interfaces podem ser classificadas em dispositivos de entrada e de saída. Os de entrada são dispositivos que os participantes utilizam para enviar algum sinal para o AV, por exemplo, o teclado. E os dispositivos de saída são aqueles que o AV envia alguma informação para o participante, como o monitor, por exemplo.

Normalmente, a visão é o sentido mais explorado em sistemas de RV. Além de monitores com alta resolução, projeções em paredes e capacetes de RV proporcionam maior grau de imersão. A CAVE (*Cave Automatic Virtual Environment*) desenvolvida inicialmente na Universidade de Illinois, Chicago, em 1992, tornou-se bastante popular pelas suas características de imersão, tamanho real e visualização em grupo (CRUZ-NEIRA *et al*., 1992). O ambiente é baseado em um cômodo, onde as paredes, piso e teto são telas que recebem projeções sincronizadas das partes de um AV. O DOMO também é um dispositivo de RV similar à CAVE. No entanto, possui um único projetor, dotado de uma lente especial, capaz de produzir uma imagem relativamente pouco distorcida, com uma superfície equivalente a um quarto de esfera. Já a CAVE é considerada mais sofisticada em relação ao DOMO, pois conta com diversos projetores, proporcionando maior grau de imersão (DE AMEILDA & LARA, 1999). A Figura 2.2 (a) mostra um DOMO em formato oval e na Figura 2.2 (b) mostra algumas pessoas assistindo as visualizações dentro do DOMO. Os participantes também podem controlar a navegação por meio de rastreadores e utilizar óculos estereoscópicos para transmitir profundidade dos objetos virtuais (TORI & KIRNER, 2006), conforme ilustra a Figura 2.2 (c).

Os primeiros protótipos que isolavam totalmente o participante do mundo externo eram capacetes com displays visuais e sonoros acoplados, conhecidos como

Head-Mounted Display (HMD). Atualmente, com os avanços tecnológicos, os HMDs ou os óculos de RV estão mais leves e menores. Este dispositivo funciona como pequenos monitores que exibem imagens para cada olho, proporcionando estereoscopia e isolando o participante do mundo real (TORI & KIRNER, 2006). Alguns óculos de RV estão disponíveis no mercado, tais como Oculus Rift (OCULUS, 2016), HTC Vive (HTC, 2016) e PlayStation VR (PLAYSTATION, 2016). A Figura 2.3 mostra o kit do Oculus Rift para consumidores. Além dos óculos, o kit é composto por um sensor de movimento, um controle para *games* e um controle de interações básico. A Figura 2.4 mostra um exemplo de utilização do HMD e do controle básico do Oculus Rift.

(a) (b)

(c)

Figura 2.2 – Exemplo de um DOMO

Adicionalmente, é possível transformar um *smartphone* em óculos de RV utilizando dispositivos especiais, tais como Google Cardboard (GOOGLE, 2016) e Samsung Gear VR (SAMSUNG, 2016). A Figura 2.5 mostra o Google Cardboard, conhecido como os óculos de Realidade Virtual da Google. Nestes HMDs móveis, as imagens para cada olho são separadas no visor do *smartphone* e quando acoplado ao

dispositivo, as lentes biconvexas refletem as imagens ao olho humano (PARSONS & COBB, 2011).

Figura 2.3 – Oculus Rift versão para consumidor

Figura 2.4 – Exemplo de interação com RV

Figura 2.5 – Google Cardboard[2]

Uma maneira intuitiva de comandar ações no mundo virtual é a utilização de gestos capturados por luvas e sensores de gestos. Uma luva é construída com material leve, usando transdutores acoplados ao longo dos dedos, ou seja, convertendo um estímulo em uma resposta. Um rastreador no pulso fornece o posicionamento e a orientação da mão, enquanto os transdutores dão os movimentos dos dedos. Uma variação das luvas são aquelas com reação de força, constituídas de sensores e atuadores, dando a impressão de toque real nos objetos (TORI & KIRNER, 2016). O Kinect (MICROSOFT, 2016) e o Leap Motion (LEAP MOTION, 2016) são exemplos de dispositivos que capturam o movimento do corpo ou alguma parte específica do corpo através de seus sensores. A Figura 2.6 mostra a utilização do Leap Motion para interagir com imagens no computador. O dispositivo posicionado próximo ao teclado, captura os movimentos das mãos e atualiza a posição da imagem de acordo com os gestos realizados.

Além de criar ambientes de RV apenas com imagens geradas através de computador, tem-se utilizado imagens reais gravadas com câmeras que capturam em 360º. Com este tipo de dispositivo, além de fotografar, é possível gravar vídeos. A Figura 2.7 mostra uma câmera 360º.

[2] https://vr.google.com/cardboard/

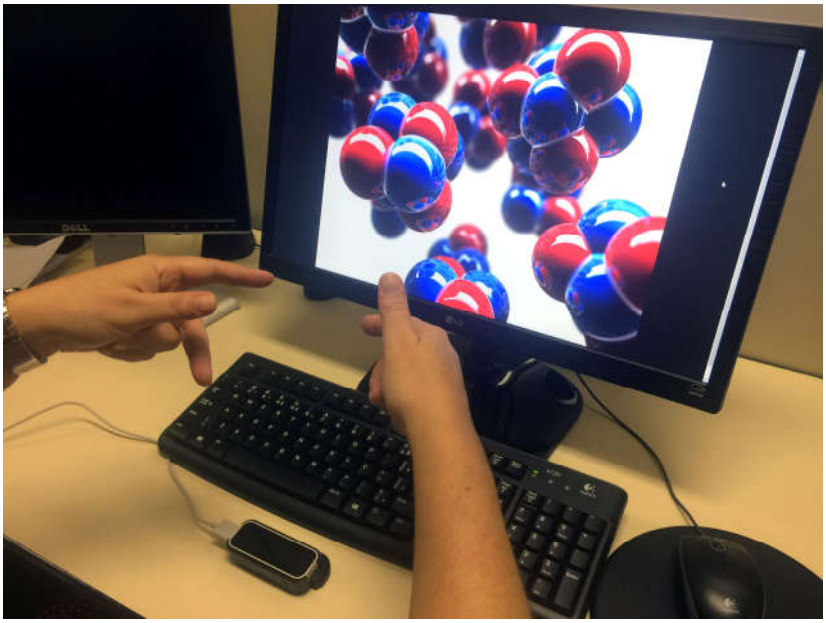


Figura 2.6 – Utilização do Leap Motion

Figura 2.7 – Câmera 360º

2.2.5. Participantes

Tal como acontece em outros sistemas computacionais, um usuário ou participante, é um elemento essencial em qualquer experiência de RV. Segundo Muhanna (2015), é importante levar em consideração três tipos de participantes no desenvolvimento de sistemas de RV: os participantes novatos, os quais precisam de elementos para apoiar na aprendizagem e interação com o AV; os especialistas que, por outro lado, necessitam de um sistema eficiente para alcançar o objetivo por meio de

atalhos e comandos especiais; e os ocasionais, que necessitam de uma combinação de recursos baseados nos dois tipos anteriores.

Além disso, uma relação entre um sistema de RV e seu participante pode assumir a forma de um-para-um ou um-para-muitos. Numa relação de um-para-um, apenas um participante interage com o AV. Uma relação de um-para-muitos, por outro lado, permite que muitos participantes interajam com o AV, ou seja, ambientes virtuais colaborativos que suportam participantes simultâneos em seu ambiente. É importante notar que no mínimo um participante deve estar interagindo com o AV, enquanto outros participantes observam a interação. Caso contrário, o elemento de interatividade não terá efeito, resultando numa experiência fraca de RV (MUHANNA, 2015).

2.3. Visualização 3D *versus* Realidade Virtual

De maneira geral, abordagens em 3D tentam criar visualizações, as quais são próximas das metáforas do mundo real ou melhoram o uso do espaço em adicionar mais uma dimensão. O usuário é capaz de rotacionar e mover objetos 3D e navegar dentro do mundo tridimensional. Algumas abordagens propõem usar um layout 2D visto sobre uma perspectiva 3D com interação limitada para o 2D, o qual pode ser denominada uma abordagem 2.5D (TEYSEYRE & CAMPO, 2009).

Pesquisas recentes mostram que representações em 2D e 3D são úteis para diferentes tipos de tarefas, e por isso, combinações das duas formas são aceitas (TORY *et al.*, 2006). Por outro lado, a questão dos benefícios oferecidos pelo 3D em relação ao 2D ainda permanece em aberto. De acordo com WEN (1995), quando o 2D é suficiente para mostrar as informações, não é recomendável adicionar mais uma dimensão. Esta dimensão extra deveria ser apenas usada para visualizar um conjunto de dados semanticamente mais ricos. Contudo, IRANI & WARE (2003) afirmam que representações 3D facilitam a percepção do sistema visual humano, bem como a inclusão de elementos estéticos e animação, podem aumentar o apelo do *design*, intuitividade e memorização de uma visualização.

Além disso, o 3D ajuda a ter uma percepção mais clara de relações entre objetos, na integração de visões locais com visões globais e na composição de múltiplas visões 2D em uma visão 3D (IRANI & WARE, 2003). Adicionalmente, gráficos 3D são semelhantes ao mundo real e permitem a representação de uma forma mais natural. Isto significa que a representação dos objetos pode estar relacionada com seu verdadeiro

conceito associado, as animações podem tornar imagens mais realistas e as interações podem ser mais intutitivas, como por exemplo a visualização em sistemas de RV.

De acordo com a definição adotada nesta dissertação, RV é uma interface avançada para aplicações computacionais, que permite ao usuário navegação e interação em tempo real, em um ambiente tridimensional, podendo fazer uso de dispositivos multissensoriais, para atuação ou *feedback* (TORI & KIRNER, 2006). A característica principal que pode ser extraída é a interação em tempo real com uma representação tridimensional.

Geralmente, as ferramentas de visualização de informação em 3D fornecem mecanismos que permitem ao usuário interagir com as representações. Do ponto de vista das dimensões e tempo de resposta, considera-se que abordagens de visualização em 3D podem ser consideradas visualização em RV. Como visto nas seções anteriores, sistemas de RV também fazem uso de dispositivos convencionais como nas abordagens de visualização 3D. O que é mais evidente em sistemas de RV é a exploração da imersão e a interação natural com os ambientes tridimensionais. A sensação de "estar dentro" pode ser alcançada utilizando *mouse* e um monitor comum, contudo, dispositivos que possibilitam a interação mais natural dos objetos virtuais, adiantam o processo de imersão. Outro recurso particular da RV é o uso da estereoscopia[3]. Segundo, MALETIC *et al.* (2001), visões estereoscópicas podem ajudar no entendimento de dados complexos e ambíguos, bem como apoiar na percepção de tamanho e a distância entre os objetos.

2.4. Aplicações de RV em Engenharia de Software

TEYSEYRE & CAMPO (2009) apresentam uma visão geral da área de visualização de software 3D, em que grande parte dos trabalhos propõem tipos de representações gráficas diferentes para apoiar diversas tarefas relacionadas à engenharia de software. A possibilidade de utilização de dispositivos não-convencionais, bem como a utilização da experiência imersiva, ainda é pouca explorada pelos pesquisadores em engenharia de software. A seguir são apresentados alguns trabalhos que utilizam RV para determinados fins em engenharia de software.

[3] Em computação gráfica, a estereoscopia visual é feita a partir da geração de duas imagens, a partir das localizações das câmeras virtuais separadas de uma determinada distância (TORI & KIRNER, 2006).

RADFELDER & GOGOLLA (2000) propõem uma abordagem com o objetivo de estender a UML em três e quatro dimensões, de forma que os aspectos estáticos e dinâmicos sejam dispostos em uma visão. A quarta dimensão refere-se ao comportamento dinâmico, ou seja, além das três dimensões espaciais (*x*, *y*, e *z*), a animação de diagramas é computada pelos autores como uma dimensão adicional.

MALETIC *et al.* (2001) apresentam um ambiente para a visualização de software orientado a objetos em um ambiente virtual imersivo, denominado Imsovision. O objetivo do ambiente é apoiar o entendimento de programas e o desenvolvimento por meio da visualização de software. O usuário posicionado dentro de uma CAVE, será capaz de interagir com o software orientado a objeto e explorar informações de software complexo. Além disso, foi desenvolvida uma linguagem de visualização que mapeia o código-fonte a fim de construir as visualizações no ambiente virtual, denominada COOL (Linguagem para Compreensão de Software Orientado a Objetos - tradução). Segundo os autores, esta linguagem incorpora algumas das características da UML e permite uma representação natural de um certo nível de complexidade do código-fonte.

FITTKAU *et al.* (2015) propõem melhorar a experiência do usuário na exploração de informação do software por meio de RV, utilizando dispositivos não-convencionais, tais como HMD e interação baseada por gestos. A representação gráfica do software é baseada na metáfora de cidades, onde classes são análogos aos prédios, a altura destes prédios indica a quantidade de métodos e pacotes são como quarteirões.

ELLIOTT *et al.* (2015) motivam o uso de RV para engenheiros de software, porém com o foco diferente dos trabalhos apresentados anteriormente. Os autores evidenciam os a*ffordances*[4] da RV para desenvolvedores como grande potencial de criar novas oportunidades, tais como alta produtividade, baixa curva de aprendizado, aumento na satisfação do usuário, dentre outros aspectos. O ambiente Live Coding funciona como um editor de textos para códigos, porém em um ambiente virtual. Ele

[4] *Affordance* é a qualidade de um objeto que permite ao indivíduo identificar sua funcionalidade sem a necessidade de prévia explicação, o que ocorre intuitivamente.

suporta a programação em Three.js[5] e possibilita a visualização do resultado no mesmo espaço virtual da codificação.

Nota-se que, apesar de serem poucos trabalhos, a RV é utilizada como uma nova técnica de visualização e, principalmente, na forma como usuários interagem com o software. Por meio dos dispositivos não-convencionais (CAVE, HMD, gestos, dentre outros) a possibilidade de alcançar uma experiência imersiva e explorar novos potenciais é maior em relação aos dispositivos convencionais (teclado e *mouse*).

2.5. Considerações Finais

Este capítulo teve como objetivo apresentar a Realidade Virtual como potencial tecnologia e recurso no âmbito da engenharia de software. Atualmente, grandes investimentos estão sendo realizados em aplicações e também em dispositivos de RV. Além disso, foi discutida a diferença entre visualizações 3D e RV, bem como a apresentação de alguns trabalhos que utilizam RV para apoiar a engenharia de software. Estes trabalhos não são considerados como trabalhos relacionados (Seção 3.4), pois a abordagem proposta nesta dissertação foca na análise dinâmica de sistemas de software orientado a objetos.

[5] Biblioteca em JavaScript para construção de imagens tridimensionais.

CAPÍTULO 3 - VISUALIZAÇÃO DE SOFTWARE

3.1. Introdução

O desenvolvimento de sistemas de software é uma tarefa árdua, pois envolve um conjunto de fases relacionadas ao longo do ciclo de vida do software. Durante todas essas fases, engenheiros de software precisam de maneiras diferentes de compreenderem elementos de sistemas de software de larga escala. Neste contexto, o uso de representações gráficas dos dados pode apoiar e facilitar a análise e compreensão de tais informações complexas de software (TEYSEYRE & CAMPO, 2009). Neste sentido, técnicas de visualização de software são aplicadas a fim de melhorar a compreensão dos diferentes aspectos do software e reduzir os custos de desenvolvimento (MILI & STEINER, 2002).

Segundo DIEHL (2007), a visualização de software é a arte e a ciência de gerar representações visuais dos vários aspectos do software e seu processo de desenvolvimento. Seu objetivo é ajudar a compreensão de sistemas de software e melhorar a produtividade do processo de desenvolvimento. Os aspectos do software mencionados podem ser classificados como estrutural, comportamental e evolutivo (DIEHL, 2007):

- **Estrutural**: refere-se às partes estáticas e relacionamentos do sistema, os quais podem ser computados ou obtidos sem a execução do programa. Alguns exemplos são o código-fonte, estrutura de dados, as chamadas estáticas entre objetos e a organização do programa em módulos;
- **Comportamental**: este aspecto está relacionado com a execução do software, ou seja, propriedades do software que são obtidas exclusivamente quando o software é executado. Dependendo da situação, a execução pode ser visualizada em alto nível de abstração, como funções chamando outras funções ou objetos se relacionando com outros objetos;
- **Evolutivo**: trata do processo de desenvolvimento do software e, em particular, enfatiza a mudança dos artefatos de software ao longo do tempo, como por exemplo, alterando funcionalidades do código-fonte do sistema ou simplesmente removendo bugs.

O foco desta dissertação concentra-se na visualização do comportamento dinâmico do software. Na literatura técnica, o termo análise dinâmica refere-se ao campo de pesquisa que trata sobre a análise de propriedades da execução do software. Portanto, este trabalho usará o termo análise dinâmica como referência ao comportamento dinâmico do software.

Além de apresentar uma visão geral sobre a área de visualização de software nesta seção, o restante deste capítulo irá abordar outros assuntos, organizados como segue. A Seção 3.2 apresenta as dimensões de visualização de software; a Seção 3.3 discute o conceito de análise dinâmica e o uso de visualização para apoiar a compreensão do comportamento dinâmico de software, bem como os trabalhos relacionados; a Seção 3.4 faz uma análise sobre trabalhos relacionados, enquanto que a Seção 3.5 finaliza o capítulo apresentando as considerações finais.

3.2. Dimensões da Visualização

MALETIC *et al.* (2002) definiram um framework com cinco dimensões com o objetivo de descrever uma gama de ferramentas de visualização que apoiam atividades de engenharia de sistemas de software de larga escala. As dimensões englobam as classificações propostas por taxonomias de visualizações muito referenciadas (PRICE *et al.* (1993); ROMAN & COX (1993)) e correspondem ao *porquê*, *quem*, *o que*, *como* e *onde* de uma visualização, as quais serão descritas nas seções seguintes.

3.2.1. Tarefas: por que a visualização é necessária?

A dimensão tarefa define por que a visualização é necessária. Em outras palavras, esta dimensão especifica quais tarefas de engenharia serão apoiadas pela visualização. Em geral, cada sistema de visualização apoia a compreensão de um ou mais aspectos de um sistema de software. Este processo de compreensão, por sua vez, apoia uma tarefa em particular. Muitas das ferramentas de visualização apoiam sistemas complexos nas atividades de desenvolvimento, na manutenção e evolução, engenharia reversa, gerenciamento de projetos de software, dentre outras. O desenvolvedor exigirá diferentes ferramentas de visualização, cada uma com suas características, devido à necessidade de obter a compreensão de diferentes níveis de abstração do sistema.

3.2.2. Audiência: quem utilizará a visualização?

Com base na tarefa apoiada, a ferramenta de visualização de software pode ser orientada a diferentes tipos de usuários. Esta dimensão define qual audiência será

atendida pelo sistema de visualização. Por exemplo, se a tarefa principal for para ensino-aprendizagem, estudantes e professores formam a audiência, bem como usuários que atuam no desenvolvimento, manutenção e evolução, nos testes de software, na gerência do projeto, dentre outras atividades.

3.2.3. Alvo: quais os dados que serão representados?

O objetivo de um sistema de visualização de software define quais aspectos do software serão visualizados, ou seja, o produto de trabalho, artefato, ou parte do ambiente do sistema de software. Exemplos incluem arquitetura, *design*, algoritmo, código-fonte, dados, rotas de execução etc. Outro tipo de dados de origem é a métrica de software, que extrai informação complementar podendo ser utilizada como medida de acordo com seu contexto. Este tipo de informação apoia a compreensão em diversas atividades de engenharia de software.

Considerando que os sistemas de software atuais são complexos, pois possuem milhões de linhas de código, um aspecto muito importante a ser tratado é o problema de escalabilidade. Para representar o grande volume de dados de sistemas complexos são necessárias mídias especiais e técnicas de representação adequadas ao tipo de complexidade.

3.2.4. Representação: como serão representados?

Dependendo dos objetivos do sistema de visualização de software, o tipo de audiência e meio disponível, uma forma de representação precisa ser definida para melhor transmitir as informações ao usuário. Esta dimensão define como a visualização é construída com base na informação disponível. Na concepção de um sistema de visualização de software, este é um dos elementos mais importantes.

Mackinlay (1986) definiu dois critérios para avaliar o mapeamento de dados a uma metáfora visual: expressividade e eficácia. Estes critérios foram utilizados no mapeamento 2D, mas também podem ser aplicados para mapeamentos em 3D.

A expressividade refere-se à capacidade de a metáfora representar visualmente todas as informações que deseja visualizar. Por exemplo, se o número de parâmetros visuais disponíveis na metáfora para a exibição de informações é menor do que o número de valores de dados que deseja visualizar, a metáfora não será capaz de cumprir o critério de expressividade.

O segundo critério, refere-se à eficácia da metáfora em melhor representar a informação, preocupando-se em alguns aspectos, tais como conceitos estéticos, otimização de imagens, dentre outros. Em relação aos dados quantitativos, como métricas, rotas de execução e número de linhas de código, além da parametrização visual corresponder aos dados, o mapeamento deve ser realizado com os dados certos. A eficácia implica a categorização de parâmetros visuais em relação à capacidade de codificar tipos diferentes de informação. Além disso, também implica na categorização de informação de acordo com sua importância, para que a informação, a qual possui um alto nível de relevância, possa ser codificada de forma eficaz.

Também é importante considerar a semântica, simplicidade e nível de abstração das visualizações. O objetivo da metáfora ou linguagem visual é poder representar uma variedade de elementos sem ambiguidades ou sem perda de significado. Além disso, a visualização deve maximizar o potencial da mídia utilizada. Por exemplo, uma boa representação em RV pode fazer uso de todas as possibilidades de navegação em um ambiente virtual tridimensional, transmitindo a sensação de imersão ao usuário, enquanto interage com a representação de forma natural e evitando sobrecarga de informação.

3.2.5. Meio: onde a representação será visualizada?

Esta dimensão diz respeito ao meio pelo qual a visualização é processada, ou seja, a mídia utilizada para a exibição de imagens e interação do usuário com a ferramenta de visualização. Os meios mais utilizados e conhecidos são monitores, variando em resolução, tamanho em polegadas e cores, bem como o uso de dispositivos de entrada, como teclado e *mouse*. O uso de projetores e múltiplos monitores aumentam o tamanho da tela, possibilitando a visualização de grandes estruturas ou muitas informações dispersas, mas que de alguma forma possui algum relacionamento.

O objetivo desta dimensão é explorar o potencial de novas mídias aplicadas à exploração e navegação de informação, principalmente sobre dados de sistemas de software de larga escala. Atualmente, pesquisas e investimentos por parte da indústria estão contribuindo para o avanço e popularização da experiência de interação com ambientes virtuais imersivos. A RV possibilita a interação mais natural com simulações e sistemas computacionais. Por exemplo, utilizando óculos de RV com imagens estereoscópicas e com sensores de gestos, o usuário pode ter a sensação de “estar

dentro" do ambiente virtual. Deste modo, as mesmas interações no mundo real podem ser realizadas também no virtual, tais como andar para obter outros pontos de vista, interagir e "pegar" objetos virtuais, dentre outros. Outros sentidos humanos como tato e olfato, por exemplo, podem ser utilizados em visualização de informação em ambientes virtuais.

3.3. Visualização aplicada à Análise Dinâmica

Esta seção tem como objetivo apresentar o conceito de análise dinâmica, bem como o uso de visualização para apoiar a compreensão do comportamento dinâmico de software. As abordagens propostas na literatura técnica, tratadas nesta dissertação como trabalhos relacionados, serão apresentadas de acordo com o tipo de representação gráfica: bidimensional (Seção 3.3.2) e tridimensional (Seção 3.3.3).

3.3.1. Análise Dinâmica

Um dos aspectos mais importantes na manutenção de software é compreendê-lo adequadamente. Em particular, entender sistemas de software de larga escala envolve um grande esforço mental por parte dos mantenedores, devido à complexidade inerente do sistema, a qual é distribuída ao longo de vários tipos de artefatos. Além disso, este processo de compreensão pode representar até 60% da manutenção, podendo impactar negativamente no tempo e no custo do projeto (CORBI, 1989; FJELDSTAD & HAMLEN, 1979).

Considerando que a compreensão de programas é custosa, o desenvolvimento de técnicas e ferramentas que apoiam esta atividade pode influenciar na eficiência global do desenvolvimento de software. A literatura oferece muitas dessas técnicas: exemplos incluem a análise de rotas de execução, redescoberta da arquitetura, e localização de características (uma atividade que envolve o relacionamento de requisitos com o código-fonte). A maioria das abordagens podem ser divididas em análise estática e dinâmica e suas combinações (CORNELISSEN, 2009).

O foco desta dissertação é a análise dinâmica, a qual investiga aspectos comportamentais de um sistema de software. BALL (1999) define análise dinâmica como a análise de propriedades da execução de um sistema de software. A Figura 3.1 mostra uma visão geral dos principais passos da análise dinâmica. Normalmente, a captura dos dados do software durante a execução é por meio de interpretação (por exemplo, usando a máquina virtual em Java) ou instrumentação (por exemplo, usando

AspectJ (KICZALES *et al*., 2001)). Os dados resultantes podem ser utilizados para fins de engenharia reversa, depuração e, por muitas vezes, para a análise de rotas de execução (BENNETT *et al.*, 2008). Na literatura técnica, foram propostas várias abordagens de análise dinâmica para apoiar a compreensão de programas, resultando em diferentes técnicas e ferramentas.

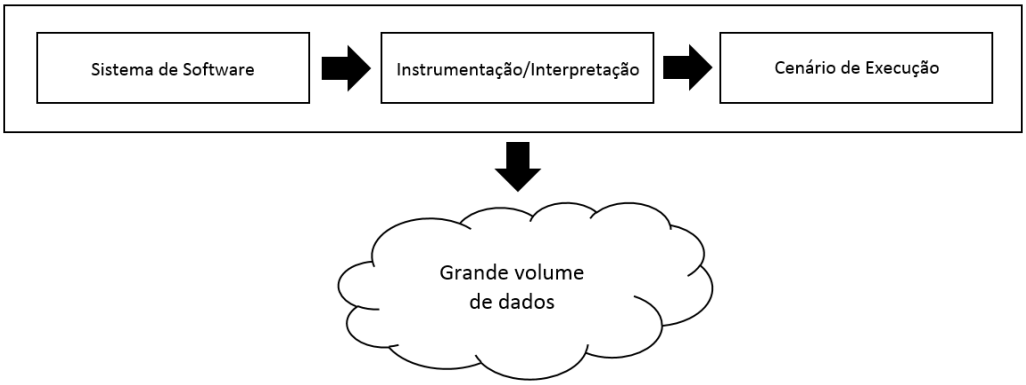


Figura 3.1 – Principais passos da análise dinâmica

CORNELISSEN *et al.* (2009) descrevem alguns benefícios e limitações da análise dinâmica para a compreensão de programas. As vantagens são:

- A *precisão* no que diz respeito ao comportamento real do sistema de software, por exemplo, na identificação de polimorfismo e na ligação dinâmica, no contexto de programas orientados a objetos (BALL, 1999);
- O fato de uma *estratégia orientada ao objetivo* poder ser utilizada, o que implica a definição de um cenário de execução, de tal forma que apenas as partes de interesse do sistema de software são analisadas.

Dentre as limitações da análise dinâmica, destacam-se:

- Sua *incompletude*, no sentido de que os dados capturados da execução do software são altamente dependentes do escopo e dos estímulos de entrada recebidos. Sendo assim, não há garantia da cobertura total das características do sistema, ou seja, objetos e mensagens podem ser instanciados ou não, dependendo do cenário definido (BALL, 1999). Note-se que a mesma limitação se aplica a testes de software.
- A dificuldade de determinar quais *cenários* executar, a fim de acionar os elementos do programa de interesse. Na prática, conjuntos de testes podem ser

usados ou execuções registradas envolvendo a interação do usuário com o sistema (BALL, 1999).

- O efeito de *observação*, isto é, o fenômeno no qual o software age de forma diferente quando está sob observação, pode representar um problema em programas *multithreaded* e sistemas de tempo real (ANDREWS, 1998).
- A *escalabilidade* da análise dinâmica devido ao grande volume de dados que pode interferir no desempenho dos sistemas, armazenamento (ZAIDMAN, 2006) e, sobretudo, em sobrecarga cognitiva (ZAYOUR & LETHBRIDGE, 2001).

Em relação à escalabilidade da análise dinâmica, técnicas de visualização têm sido aplicadas, a fim de melhor transmitir informações complexas da execução e reduzir a sobrecarga cognitiva. Nas seções a seguir, serão apresentadas as técnicas de visualização aplicadas à análise dinâmica, porém sob a perspectiva do tipo de representação gráfica.

3.3.2. Representação Bidimensional (2D)

Nesta seção, serão apresentados trabalhos considerados relacionados com a abordagem proposta nesta dissertação, porém, a representação gráfica dos dados é construída em duas dimensões.

3.3.2.1. SHIMBA

SHIMBA é uma ferramenta para a compreensão do comportamento de sistemas escritos na linguagem Java (SYSTÄ *et al.*, 2001). Os desenvolvedores escolhem explicitamente os artefatos que desejam ser rastreados (por exemplo, classes, interfaces, métodos etc.). Para a visualização do rastreamento, SHIMBA usa SCED, um *framework* de visualização de rotas de execução para criar e manipular diagramas de sequência e máquinas de estados semelhantes aos diagramas UML e é capaz de detectar sequências repetidas de chamadas idênticas. O propósito da ferramenta SHIMBA não é fornecer visões sobre o comportamento geral de um sistema de software, mas sobre o comportamento de artefatos selecionados manualmente. Deste modo, os rastros de execução não serão muito grandes, reduzindo a possibilidade de gerar problemas de escalabilidade.

3.3.2.2. ISVIS

ISVIS (JERDING & STASKO, 1998) introduz o conceito de visualização de mural de informações para representar as rotas de execução de sistemas de software de larga escala orientados a objetos de forma condensada (Figura 3.2). Os objetos ocupam linhas verticais e a cor é adicionada ao longo do tempo quando as mensagens são enviadas. Além desta visão, é proporcionada uma representação semelhante a um diagrama de sequência UML. Nesta visão, um algoritmo de detecção de padrões verifica sequências de chamada idênticas.

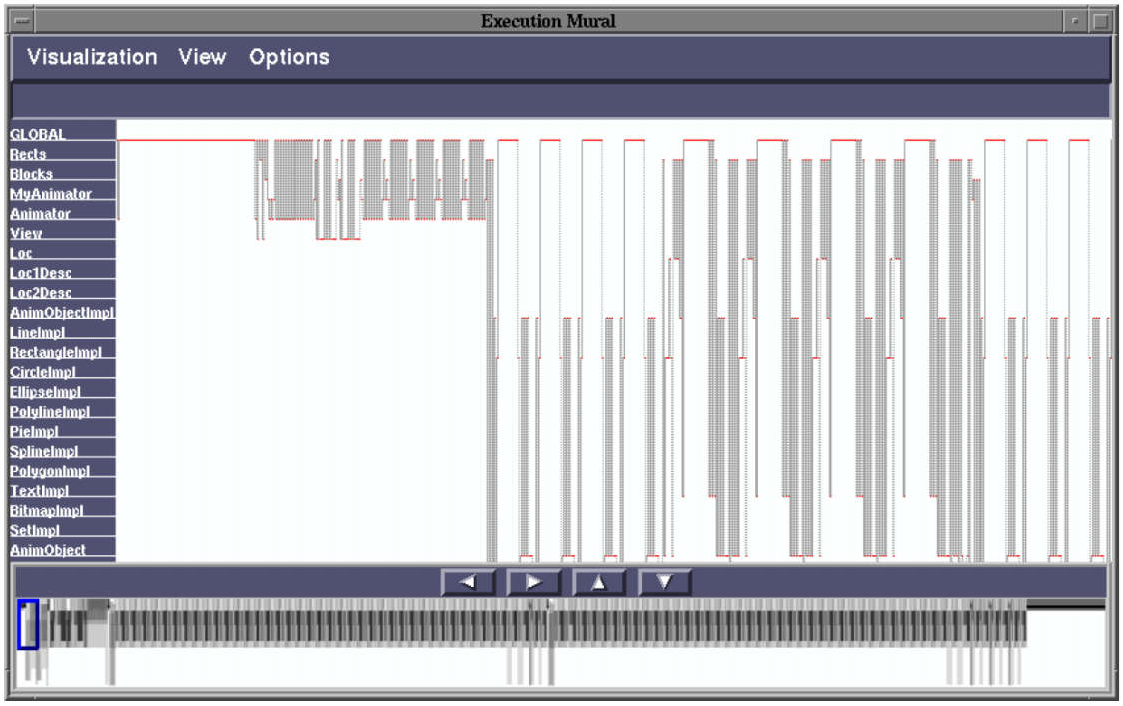


Figura 3.2 – Ferramenta ISVIS (JERDING & STASKO, 1998)

3.3.2.3. OVATION

DE PAUW *et al.* (1998) desenvolveram a OVATION, uma ferramenta para apoiar a detecção de padrões de execução de sistemas orientados a objetos por meio de visualização e exploração do sistema em vários níveis de abstração (Figura 3.3). A ferramenta emprega técnicas visuais, de navegação e analíticas que suportam os longos rastros de execução. Ao identificar o comportamento repetido, como por exemplo, chamadas de métodos idênticos, as informações são agrupadas, reduzindo as informações de execução que os programadores devem assimilar, com pouca perda de percepção.

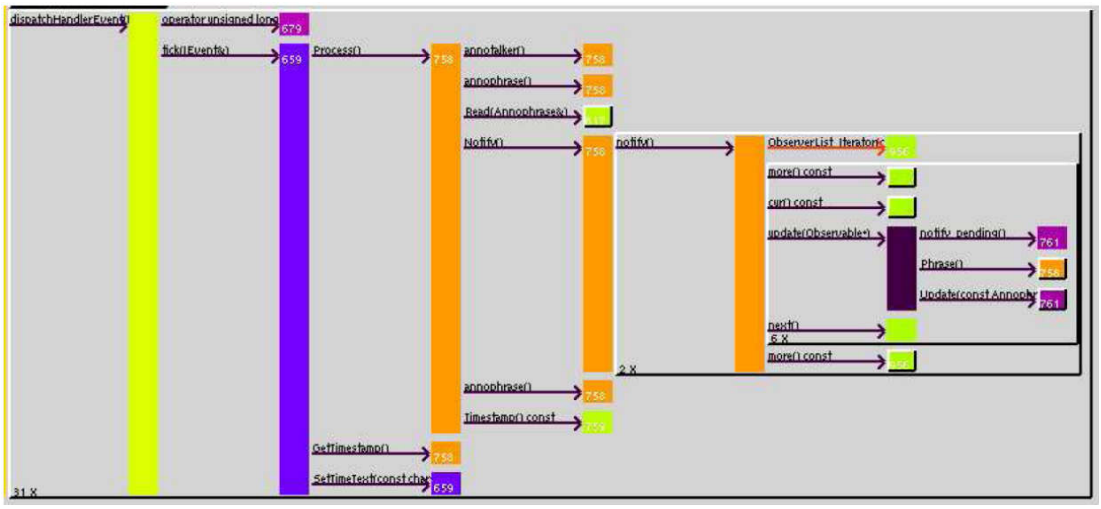


Figura 3.3 – Ferramenta Ovation (PAUW *et al.*, 1998)

3.3.2.4. AVID

AVID é uma ferramenta de visualização de rotas de execução para sistemas na linguagem Smalltalk (WALKER *et al.*, 1998). Os dados de rastreamento são obtidos através da instrumentação da máquina virtual Smalltalk. AVID cria visualizações que mostram principalmente a estrutura modular do sistema. Sobre a visualização estrutural, a informação dinâmica contida em um rastro é sobreposta (Figura 3.4).

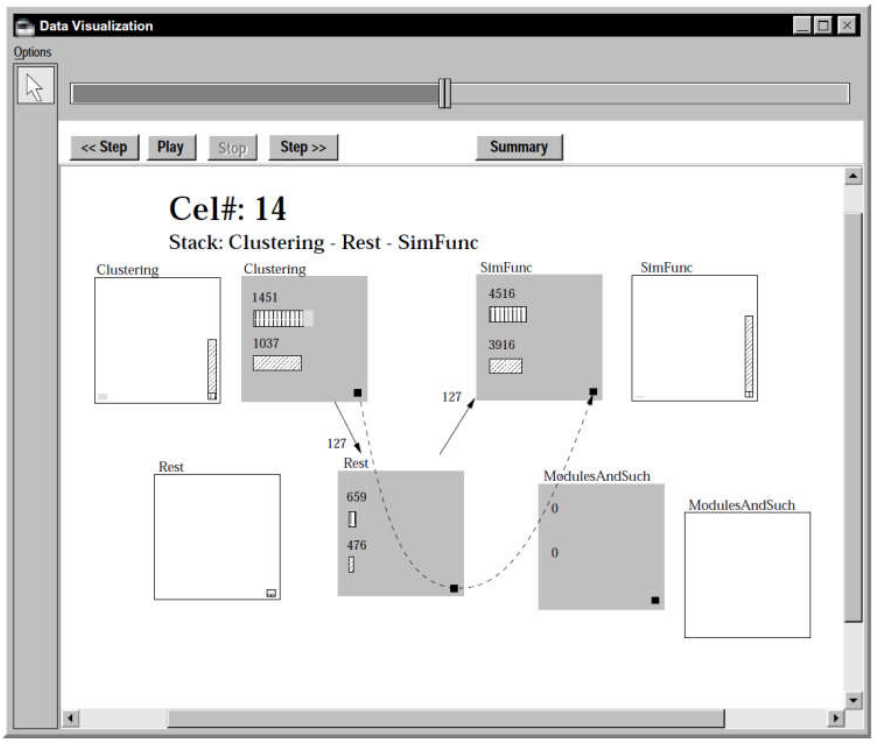


Figura 3.4 – Ferramenta AVID (WALKER *et al.*, 1998)

3.3.2.5. EXTRAVIS

EXTRAVIS (CORNELISSEN *et al.*, 2008) utiliza duas visualizações sincronizadas que permitem aos desenvolvedores explorarem o rastreamento de programas (Figura 3.5). A metáfora visual abordada é denominada como visão massiva de sequências, a qual retrata a chamada de métodos de uma maneira ordenada pelo tempo ao longo da dimensão vertical. A visão de pacote circular representa os elementos estruturais do sistema na forma de um círculo, incluindo sua estruturação hierárquica. Dentro do círculo, as lacunas entre os elementos estruturais na circunferência representam relações de chamada. Um rastro de execução é explorado selecionando um intervalo de tempo na visão massiva de sequências, analisando as relações de chamadas que estão ativas dentro do intervalo de tempo.

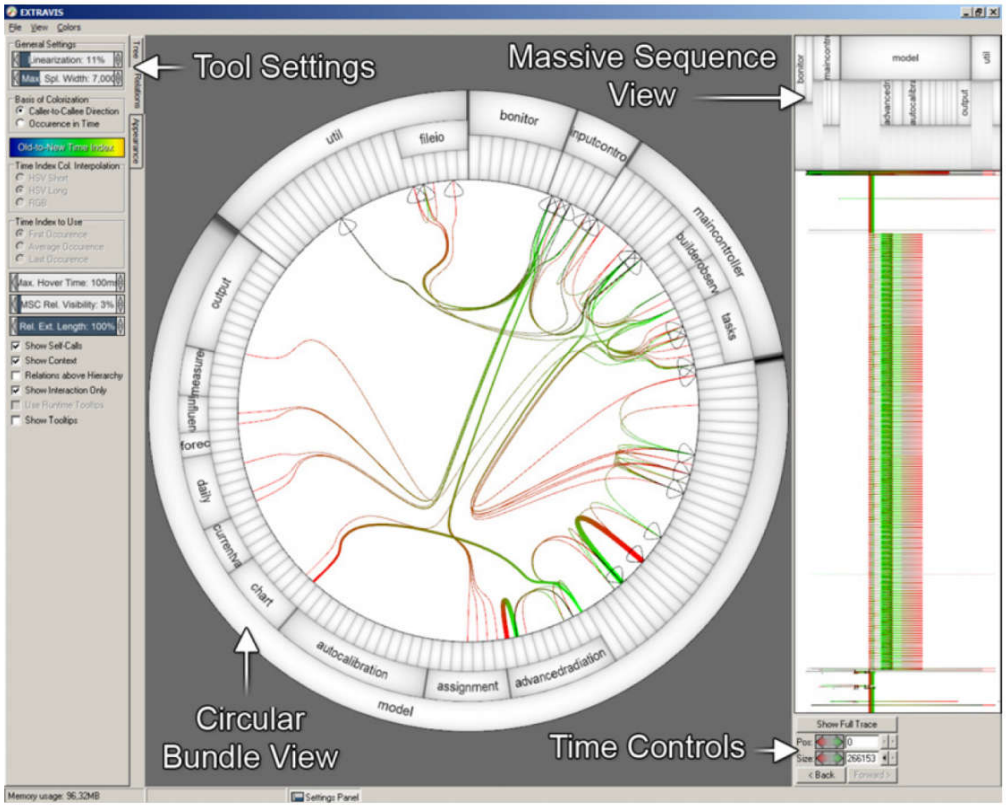


Figura 3.5 – Ferramenta EXTRAVIS (CORNELISSEN *et al.*, 2008)

3.3.2.6. ZEST SEQUENCE VIEWER

A ZEST SEQUENCE VIEWER (BENNETT *et al.*, 2007) fornece 3 visões relacionadas aos rastros de execução de sistemas em Java (Figura 3.6): um diagrama de sequência UML; uma visão miniaturizada das informações; e uma visão de diagrama de

classes UML que mostra as dependências estruturais entre os objetos relacionadas ao rastro de execução.

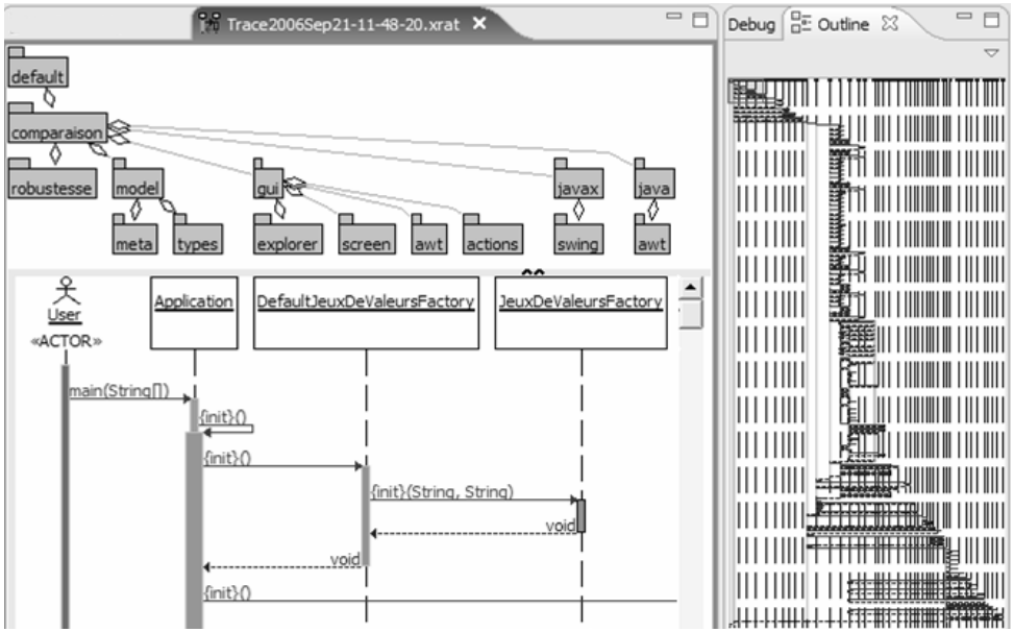


Figura 3.6 – Ferramenta Zest Sequence Viewer (BENNETT *et al.* 2007)

3.3.3. Representação Tridimensional (3D)

A partir desta seção são apresentados os trabalhos relacionados que utilizam a representação tridimensional.

3.3.3.1. EVOSPACES

A ferramenta EVOSPACES explora a metáfora analógica à cidade, representando o software com distritos e edíficos em um ambiente virtual 3D (DUGERDIL & ALAM, 2008). Além de representar a arquitetura, é possível obter informações de execução por meio da visão noturna. Deste modo, a visão diurna informa sobre a arquitetura, enquanto que a noturna corresponde ao comportamento dinâmico do sistema.

Nesta metáfora de cidade, cada distrito corresponde a um pacote e os edifícios são as classes pertencentes aos seus respectivos pacotes. Nos edíficos foram aplicadas duas métricas: a quantidade de linhas de código para informar o tipo e a quantidade de métodos para determinar a altura. Além disso, os métodos de cada classe são pessoas dentro dos edifícios. As relações entre as classes são representadas por linhas sólidas e a navegabilidade do fluxo das informações são representadas por um segmento vermelho. A Figura 3.7 mostra a visão arquitetural da EVOSPACES.

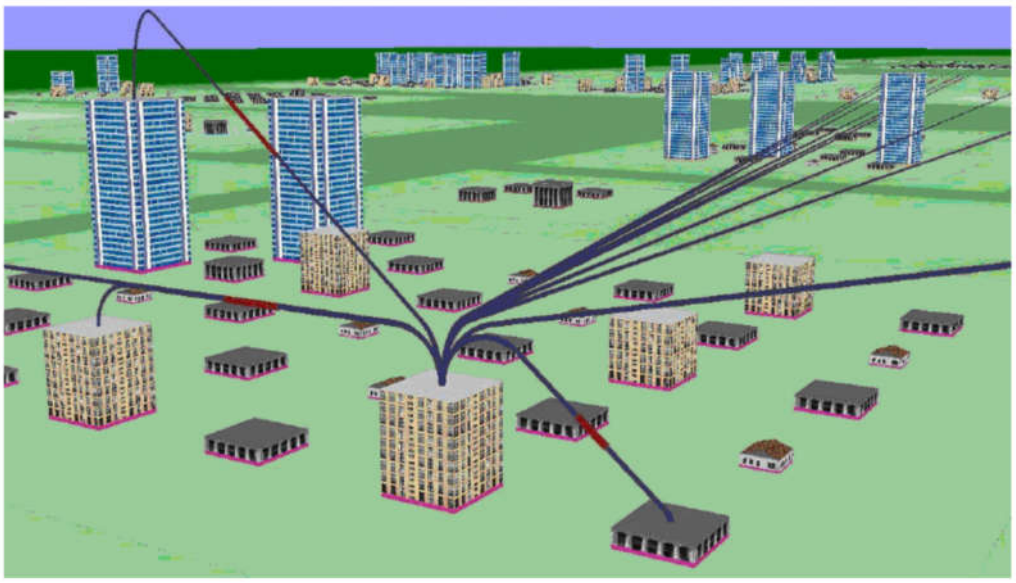


Figura 3.7 – Visão diurna da EVOSPACES (DUGERDIL & ALAM, 2008)

A visão noturna exibe as informações dinâmicas relacionadas a um cenário de execução (Figura 3.8). Um edifício é iluminado se a classe que o representa estiver envolvida na execução em um determinado segmento. Para cada segmento, os edifícios são coloridos dependendo do número de ocorrências da classe no segmento atual. Em seguida, exibindo cada segmento em uma seqüência, o usuário visualiza dinamicamente quais classes são usadas durante a execução do sistema.

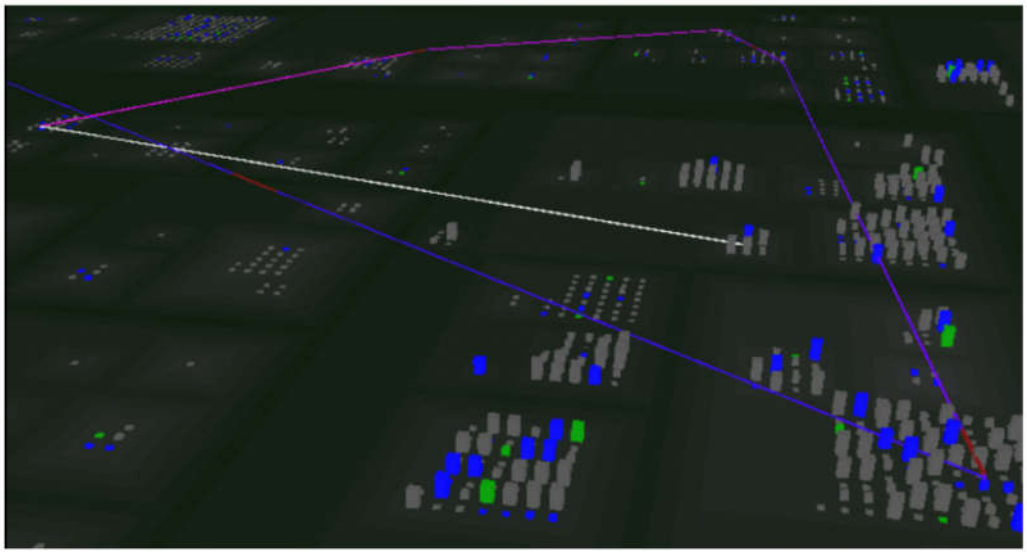


Figura 3.8 – Visão noturna da EVOSPACES (DUGERDIL & ALAM, 2008)

3.3.3.2. 3D-PP

A fim de ajudar programadores na depuração visual, OSHIBA & TANAKA (1999) propuseram um método para visualizar a transição de estados, bem como a

execução do programa com animação em um ambiente tridimensional, denominado 3D-PP. O 3D-PP é um grafo hierárquico tridimensional composto por nós e arestas. Nós correspondem aos dados, operadores e processos. O formato de um nó indica seu tipo. Por exemplo: esferas representam inteiros e *strings*; cones invertidos representam operadores, tais como adição, módulo, dentre outros; e pilares representam processos. Uma aresta corresponde à dependência de dados.

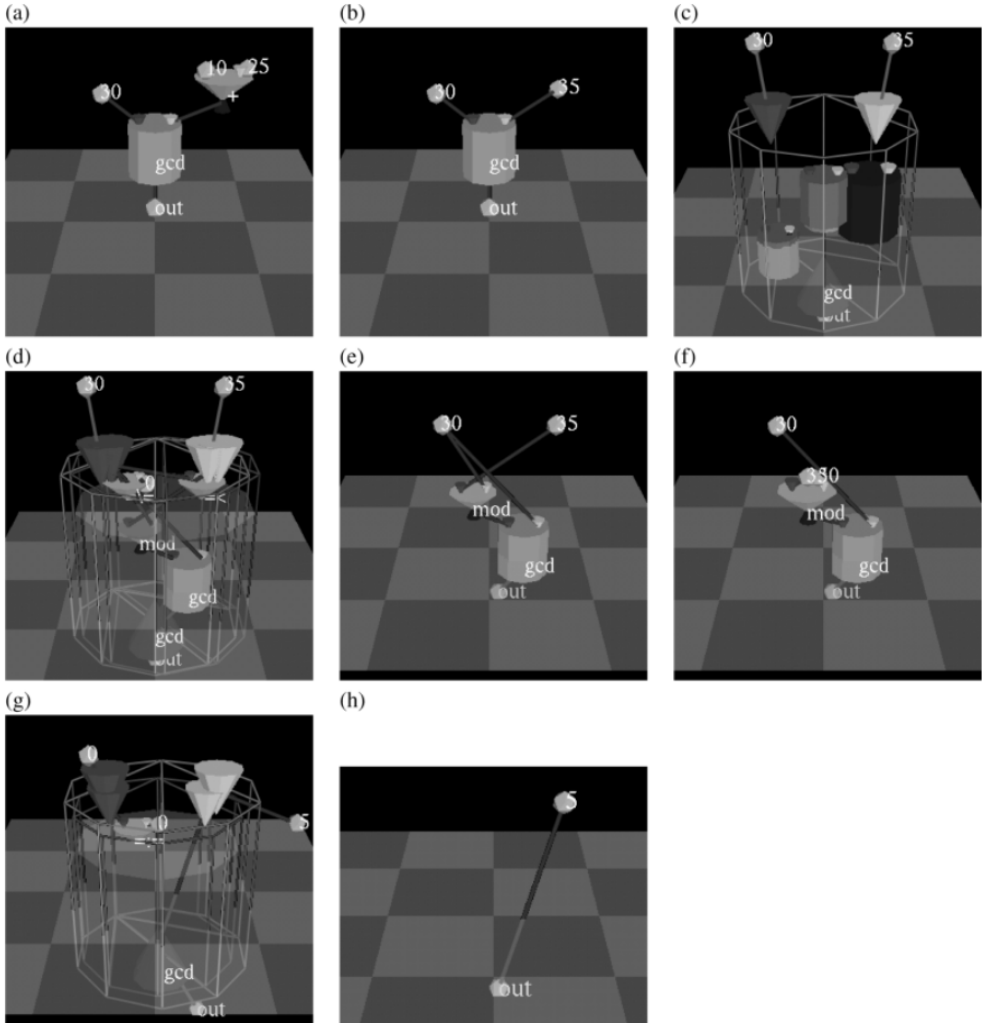


Figura 3.9 – Visualização proposta por OSHIBA & TANAKA (1999)

A Figura 3.9 (a) mostra um exemplo de utilização. O programa adiciona os números 10 e 25, bem como a utilização de um processo rotulado como "gcd" que recebe como argumentos os valores 30 e o resultado da adição. O resultado do processo é atribuído à variável "out". Conforme o programa é executado, os valores são atualizados, bem como a representação gráfica. A partir da Figura 3.9 (b) são demonstrados o passo a passo da execução com animação. Além de mostrar os argumentos e saídas, a visualização também retrata o funcionamento interno do

processo, preservando a forma geométrica somente com bordas e sem preenchimento, como mostra a Figura 3.9 (c). Ao longo da Figura 3.9 são retratados todos os passos executados do programa com animação até chegar ao final do processo que atribui o valor 5 à variável “out”, finalizando a animação.

O ambiente permite que o programador controle a execução animada por meio de algumas funções: a animação pode ser interrompida e retomada de onde parou; é possível voltar a animação de qualquer ponto da execução; a velocidade da animação pode ser alterada; e a mudança dos pontos de vista para visão panorâmica, controle do *zoom* e da rotação, além de ser possível mover objetos, como dados e processos, para examinar objetos obstruídos por outros.

3.3.3.3. EXPLORVIZ

FITTKAU *et al.* (2013) propõem uma combinação de visões estáticas e dinâmicas para apoiar o processo de compreensão de software de larga escala. A abordagem EXPLORVIZ monitora a execução do programa e disponibiliza as informações através de duas perspectivas: visão panorâmica e nível de sistema.

A visão panorâmica fornece a interação de diferentes nós, utilizando a familiaridade com diagramas de implantação e atividade da UML. Além disso, combina automaticamente as configurações de nós semelhantes em uma única entidade para apoiar a compreensão das aplicações existentes e suas respectivas interações. A comunicação é visualizada por linhas de uma aplicação para outra. A espessura da linha representa o número de solicitações na janela de tempo atual (Figura 3.10). Esta perspectiva de visão paronâmica detalhada é vinculada à perspectiva macro de nível de sistema.

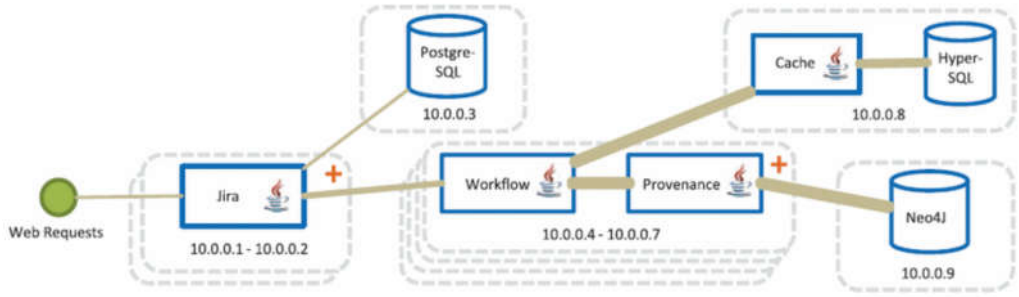


Figura 3.10 – Visão panorâmica da abordagem EXPLORVIZ (FITTKAU *et al.,* 2013)

A perspectiva nível de sistema é baseada na metáfora da cidade 3D. Conforme DUGERDIL & ALAM (2008), usam-se os conceitos comuns, tais como distritos,

edifícios e ruas. Os distritos representam componentes ou subcomponentes (por exemplo, pacotes em Java). Cada componente é visualizado como uma camada retangular com uma altura fixa. Vários componentes são empilhados uns sobre os outros para exibir suas hierarquias de subcomponentes.

Os edifícios representam entidades, isto é, componentes, subcomponentes ou classes. Na metáfora de cidade da abordagem, os edifícios se tornam distritos quando são abertos. Por exemplo, a Figura 3.11 (a) mostra o conteúdo do distrito *service* antes de ser aberta, como mostra a Figura 3.11 (b). Além disso, a largura de um edifício é determinada pelo número de classes dentro da entidade representada. Se a entidade é uma classe, a largura é um valor mínimo constante.

(a) Visão macro de quatro componentes (b) Visão de relacionamento com um componente aberto

Figura 3.11 – Metáfora de cidade da perspectiva de nível de sistema (FITTKAU *et al.,* 2013)

As ruas são representadas por tubos entre as entidades. Diferente de outras metáforas da cidade, componentes e subcomponentes podem ser parte da comunicação, além de classes. Similar à perspectiva da visão panorâmica, a espessura das ruas representa a quantidade de chamadas entre as entidades.

3.3.3.4. GREEVY *et al.* (2006)

GREEVY *et al*. (2006) desenvolveram uma nova técnica de visualização 3D para representar visualmente as instâncias de objetos e envios de mensagens, utilizando animação do comportamento, *zoom*, *panning*, rotação e detalhes sob demanda.

A Figura 3.12 mostra um exemplo do comportamento de um sistema durante a execução de uma *feature* em termos de classes, instâncias de objetos e envios de mensagens, os quais são representados por caixas. No entanto, as classes (caixas brancas) ficam na base e são ligadas por linhas que correspondem à herança. Quando ocorre a execução de uma classe é gerada mais uma caixa na cor azul, localizada acima da classe que a instanciou. Os objetos atualmente ativos são realçados em verde. Cada

vez que um objeto envia uma mensagem para outro, uma linha é desenhada entre os dois objetos. As linhas das mensagens são coloridas em vermelho para distingui-las das heranças.

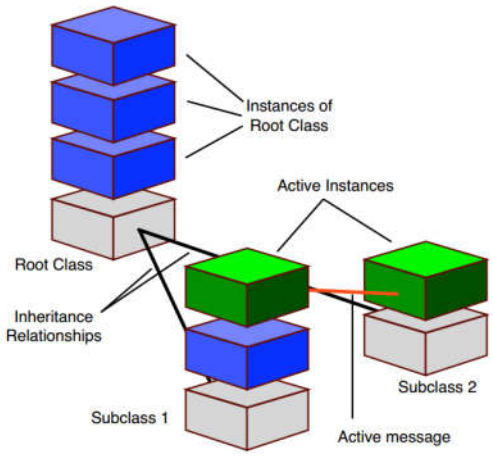


Figura 3.12 – Exemplo da visualização 3D proposta por GREEVY *et al*. (2006)

3.3.3.5. ZHAO *et al.* (2009)

O objetivo do trabalho é visualizar a comparação de múltiplas execuções do software, destacando suas semelhanças e diferenças a partir de cenários de execução distintos. São modelados visualmente execuções individuais e suas conexões com outros cenários por meio de um layout com vários planos. Cada plano é composto por um diagrama de sequência para visualizar execuções individuais.

Na representação visual proposta por ZHAO *et al.* (2009), cada objeto é representado como uma pequena esfera verde. Como cada objeto pode incluir um ou mais objetos devido à abstração, quanto mais objetos uma esfera incluir, maior será a esfera. As chamadas de método entre objetos são representadas por linhas horizontais na cor vermelha e dispostas de acordo com sua ordem de execução. O mapeamento entre as execuções é destacado com linhas amarelas, ligando os objetos dispostos nos vários planos. Os relacionamentos entre os planos são realizados quando um método é chamado por objetos da mesma classe. A Figura 3.13 (a) mostra uma típica execução, enquanto que a Figura 3.13 (b) representa várias execuções e seus relacionamentos.

Tradicionalmente, *mouse* e teclado são utilizados como meio de interação entre humanos e as visualizações através de computadores. No entanto, ZHAO *et al.* (2009) adotaram a utilização de um rastreador ocular. Com o dispositivo é possível mover, rotacionar, selecionar, ampliar e diminuir o *zoom* do ambiente e dos objetos em 3D por

meio da captura da movimentação da pupila dos olhos. Por exemplo, para selecionar um objeto e descobrir se o mesmo aparece em outras execuções, o usuário fixa o olhar durante 2 segundos. Em seguida, aplica-se o efeito de semitransparência aos elementos que não pertencem ao contexto da seleção, proporcionado um destaque visual.

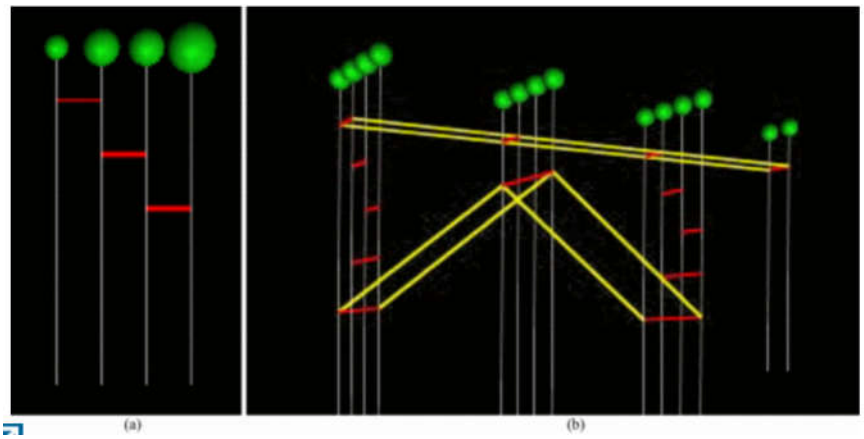


Figura 3.13 – Visualização proposta por ZHAO *et al.* (2009)

3.4. Análise dos Trabalhos Relacionados

Como apresentado neste capítulo, a visualização tem apoiado engenheiros de software à compreensão dos diversos aspectos de sistemas: estrutural, evolutivo e comportamental. Especificamente nesta dissertação, foi discutido o uso de visualização aplicada à análise dinâmica de software, categorizando-os quanto à quantidade de dimensões gráficas, ou seja, duas e três dimensões. Algumas discussões acerca dos trabalhos relacionados são apresentadas a seguir.

Grande parte dos trabalhos representam graficamente os aspectos estáticos e dinâmicos do software. Além disso, essas informações podem ser apresentadas em uma visão ou mais de duas visões. Por exemplo, a maioria das abordagens em 3D aproveitam a terceira dimensão para unir informações estruturais e dinâmicas na mesma representação gráfica, como é o caso da EVOSPACES, EXPLORVIZ e GREEVY *et al.* (2006). Diferentemente destas, as abordagens em 2D apresentam estas informações em visões separadas, ou seja, em janelas distintas, por exemplo, ZEST SEQUENCE VIEWER e EXTRAVIS. AVID utiliza uma visão em sua representação gráfica bidimensional, demonstrando a execução por meio de linhas que conectam os pacotes.

Ao contrário das informações arquiteturais, a análise dinâmica lida com dados obtidos através da execução do software. Portanto, a dimensão temporal pode

influenciar na análise e na quantidade de informações adquiridas ao longo da execução. A maioria das abordagens em 3D utilizam animações para enfatizar o comportamento ao longo do tempo. Análogo a um filme, a cada intervalo de tempo são apresentadas informações, gradativamente, até chegar ao fim da execução. Nos casos das abordagens em 2D, recursos de animação não são utilizados. No entanto, algumas utilizam a quantidade de informação entre entidades durante a execução para explicitar a dinâmica entre eles. Por exemplo, EXPLORVIZ altera a espessura da linha entre componentes de acordo com o volume de dados.

Tanto 2D quanto 3D utilizam cores, a fim de sobrepor informações em suas metáforas visuais. ISVIS utiliza a cor em sua representação visual em 2D para evidenciar as mensagens enviadas. DE PAUW *et al*. (1998) utilizam as cores para indicar os lugares onde houve chamadas de métodos idênticos. No entanto, as abordagens em 3D além de usarem as cores para dar significado a alguma informação, utilizam como um aspecto estético, visto que algumas abordagens são metáforas visuais conhecidas pelos seres humanos.

Abordagens tridimensionais utilizam métricas de software para adicionar informações em suas representações gráficas. A quantidade de métodos e de linhas de código são informações bastante utilizadas pelas abordagens. De modo geral, o 3D é utilizado para criar formas visuais tridimensionais. No caso das ferramentas EVOSPACES e EXPLORVIZ, estas representam o software análogo a uma cidade, a qual é intrinsicamente tridimensional. GREEVY *et al.* (2006) usam representação polimétrica para melhorar a visualização das instâncias dos objetos sobrepostas às classes. No entanto, ZHAO *et al.* (2009) usam o 3D para conectar vários diagramas de sequência (2D) em uma única visão por meio do ambiente virtual ilimitado. Do ponto de vista de nível de abstração, a 3D-PP foi a única que trata sobre o comportamento interno de um método, enquanto que as demais, abordam a comunicação entre objetos, pacotes e módulos.

O problema da escalabilidade é um desafio muito conhecido pela área de análise dinâmica (CORNELISSEN *et al*., 2009). Como forma de resolver esse tipo de problema, alguns algoritmos foram propostos para redução do tamanho das rotas de execução e tornar a apresentação visual mais confortável para humanos. OVATION e ZHAO *et al.* (2009) utilizam a técnica de *cluster*, ou seja, por meio de seus algoritmos, as chamadas de métodos idênticas ou similares e objetos da mesma classe são

agrupadas, reduzindo a quantidade de informações a serem apresentadas. Ao contrário destas abordagens, SHIMBA fornece visões do comportamento de artefatos selecionados manualmente, portanto, os rastros de execução tendem são ser complexos.

Uma característica que se destacou em relação aos trabalhos analisados é o uso de uma interface não convencional para interagir com a visualização. Comumente, monitor, teclado e *mouse* são interfaces convencionais utilizadas na maioria das abordagens de visualização de modo geral. ZHAO *et al.* (2009) utilizaram um rastreador ocular para humanos interagirem com a visualização. A interface capta os movimentos da pupila para movimentar os objetos, bem como selecioná-los, por meio da fixação do olhar de dois segundos para o objeto.

As abordagens em 3D não exploram o ambiente virtual tridimensional e ilimitado como um novo mecanismo de interação com as diferentes perspectivas do software. Grande parte delas, criam formas tridimensionais e unificam dados estruturais e de execução em suas metáforas visuais. Além disso, nenhum trabalho faz referência ao código-fonte. Principalmente nas tarefas de compreensão durante a manutenção é primordial a localização do comportamento de uma característica implementada no código-fonte. Um dos diferenciais da abordagem proposta nesta dissertação é a flexibilidade de interação e manipulação entre as diferentes perspectivas do software, navegando do nível arquitetural, passando pelo comportamento dinâmico até chegar ao código-fonte. A Tabela 3.1 e Tabela 3.2 apresentam um quadro comparativo das abordagens em 2D e 3D descritas nesta seção, respectivamente.

3.5. Considerações Finais

Este capítulo apresentou a área de Visualização de Software à luz do *framework* de MALETIC *et al.* (2002). As dimensões tarefa, audiência, alvo, representação e meio possuem o objetivo de descrever uma gama de ferramentas de visualização que apoiam atividades de engenharia de sistemas de software de larga escala. Em consonância ao problema central desta dissertação, foram apresentados os conceitos sobre a análise dinâmica de software, seus desafios e, principalmente, a utilização de visualização como método de apoio à resolução de problemas durante a compreensão do comportamento dinâmico de sistemas de software.

Como esta dissertação de mestrado tem como objetivo propor uma nova forma de interagir com sistemas de software no contexto de compreensão do comportamento

dinâmico por meio de RV, os trabalhos encontrados na revisão *ad-hoc* da literatura foram categorizados de acordo com o tipo de representação gráfica: bidimensional e tridimensional.

Tabela 3.1 – Quadro comparativo das abordagens em 2D

Requisitos		**ABORDAGENS**					
		SHIMBA	**ISVIS**	**OVATION**	**AVID**	**EXTRAVIS**	**ZEST**
Tipo do dado	Estrutural	+	-	-	+	+	+
	Comportamental	+	+	+	+	+	+
Abstração	Redução de informação	-	+	+	-	-	-
Visão	Uma	+	+	+	-	-	-
	Múltiplas	-	-	-	+	+	+
Qualidade de software	Métricas	-	-	-	-	-	-
Código-fonte	Edição	-	-	-	-	-	-
	Visualização	-	-	-	-	-	-
Dispositivo	Convencionais	+	+	+	+	+	+
	Não-convencionais	-	-	-	-	-	-
Representação Gráfica	UML	+	-	+/-	-	-	+
	Polimétrica	-	+	+/-	+	+	-
	Metáfora	-	-	-	-	-	-
Interatividade	Recursos interativos	-	-	-	-	+	-

+ Requisito atendido - Requisito não atendido +/- Requisito parcialmente atendido

Além de construir representações tridimensionais, a RV possui outras características, tais como interação e imersão. A ideia da análise foi identificar semelhanças e diferenças entre as abordagens tanto do mesmo tipo de representação quanto de tipos diferentes.

A abordagem VisAr3D-Dynamic foi concebida baseada nos conceitos sobre RV e Análise Dinâmica apoiada por Visualização, bem como a análise dos trabalhos encontrados na revisão *ad-hoc* da literatura. Incialmente, a pesquisa focava no apoio ao ensino de modelagem (FERNANDES *et al.*, 2015), no entanto, à medida que se buscava por trabalhos de visualização, notava-se a carência no apoio ao comportamento dinâmico. No próximo capítulo, o contexto da pesquisa e a abordagem proposta serão apresentados.

Tabela 3.2 – Quadro comparativo das abordagens em 3D

Requisitos		ABORDAGENS				
		EVOSPACES	**3D-PP**	**EXPLORVIZ**	**GREEVY**	**ZHAO**
Tipo do dado	Estrutural	+	-	+	+	-
	Comportamental	+	+	+	+	+
Abstração	Redução de informação	-	-		-	+
Visão	Uma	+	+	-	+	+
	Múltiplas	-	-	+	-	-
Qualidade de software	Métricas	+	-	+	-	-
Código-fonte	Edição	-	-	-	-	-
	Visualização	-	-	-	-	-
Dispositivo	Convencionais	+	+	+	+	-
	Não-convencionais	-	-	-	-	+
Representação Gráfica	UML	-	-	-	-	+/-
	Polimétrica	-	+	+	+	-
	Metáfora	+	-	+	-	-
Interatividade	Recursos interativos	+	+	-	+	-

+ Requisito atendido - Requisito não atendido +/- Requisito parcialmente atendido

CAPÍTULO 4 - ABORDAGEM VISAR3D-DYNAMIC

4.1. Introdução

A partir da revisão da literatura descrita no Capítulo 3 -deste trabalho, foi possível observar a importância e os ganhos obtidos com o uso de visualização para apoiar a compreensão do comportamento dinâmico de software. De acordo com CASERTA & ZENDRA (2010), a visualização gráfica do software tem potencial para resultar em um entendimento melhor e mais rápido do software e suas funcionalidades, bem como seu comportamento em tempo de execução.

Neste sentido, a abordagem proposta nesta dissertação, denominada VisAr3D-Dynamic, tem como objetivo apoiar a compreensão de um grande volume de dados gerados a partir do comportamento dinâmico de software por meio de Realidade Virtual, provendo visualizações e recursos interativos que facilitem a obtenção de *insights* acerca de sistemas de larga escala.

Além desta seção introdutória, este capítulo está organizado da seguinte forma: a Seção 4.2 descreve o contexto de pesquisa no qual essa dissertação foi concebida; a Seção 4.3 apresenta uma visão geral dos principais elementos que compõem o ambiente virtual; a Seção 4.4 discute os recursos da abordagem disponíveis por meio do ambiente virtual; e, por fim, na Seção 4.5 estão as considerações finais deste capítulo.

4.2. VisAr3D

A VisAr3D (Visualização de Arquitetura de Software em 3D) é uma abordagem no contexto de ensino-aprendizagem em engenharia de software, mais precisamente na modelagem de sistemas complexos. Com a renderização dos diagramas UML, a VisAr3D propõe um ambiente virtual que facilita o manuseio destes modelos, acrescentando recursos visuais de informações que não são explícitos nos diagramas tradicionais. Além disso, sua principal ideia é organizar todo o tipo de documentação relacionada à modelagem durante o ciclo de desenvolvimento (RODRIGUES, 2012).

A abordagem baseou-se em alguns critérios para uma boa visualização 3D a fim de apoiar a compreensão, visualização e interação com o ambiente virtual. Como resultado, alguns recursos são disponibilizados, tais como visão geral, informações contextualizadas, agente de busca, ponto de vista, visões, dentre outros.

As visões possibilitam a navegação através de todos os diagramas do sistema em estudo, no qual, diferentes aspectos ou perspectivas do sistema podem ser focados de forma independente. A ideia das visões é ajudar os usuários a obterem mais informações ou detalhes sobre os diagramas. Estas visões são: Visão dos Relacionamentos com Outros Diagramas, Visão dos Relacionamentos com Outros Tipos de Diagramas, Visão de Pacote, Visão de Métrica, Visão de Atributos/Operações, Visão do Autor, Visão de Documentação, Visão de Anotação, Visão de Exercícios e Visão de Comportamento.

A Visão de Comportamento, segundo RODRIGUES (2012), tem como objetivo utilizar animações em diagramas comportamentais, possibilitando ao usuário buscar novas informações, enfatizar o raciocínio e a reflexão de forma útil, fornecendo visualizações com mais semântica, como nos casos dos diagramas de sequência e de colaboração, por exemplo. A abordagem proposta neste trabalho é um complemento da Visão de Comportamento da VisAr3D, pois adiciona recursos interativos por meio de RV com o objetivo de apoiar atividades de compreensão durante a análise dinâmica de sistemas de software.

4.3. Visão Geral

Durante a análise dinâmica de software, desenvolvedores necessitam de ferramentas que apoiem a compreensão e a redução da sobrecarga cognitiva de um grande volume de dados gerados a partir do comportamento dinâmico de software. Neste sentido, a abordagem VisAr3D-Dynamic propõe apoiar essas atividades por meio de um ambiente virtual tridimensional e interativo. Diferente das propostas documentadas na literatura técnica, a VisAr3D-Dynamic une as diferentes perspectivas do software em seu ambiente virtual.

A abordagem VisAr3D-Dynamic consiste na composição de mecanismos de renderização de modelos UML, extração de métricas, visualização de software e RV de forma integrada, permitindo a exploração e navegação de informações acerca do comportamento dinâmico de sistemas de software complexos, tendo como objetivo apoiar a compreensão de um grande volume de dados gerados a partir da execução do software.

A visão geral da abordagem é representada na Figura 4.1. Entre seus elementos, incluem-se: repositório de modelos, a API ThreeDUML, o ambiente virtual e tridimensional, mecanismos de interação e dispositivos de interação humano-

computador. A partir dos dados gerados da execução do software persistidos em arquivos XMI, o repositório de modelos armazena e organiza estes arquivos como cenários de execução. Em seguida, a API ThreeDUML realiza um mapeamento do cenário de execução, permitindo a visualização do comportamento dinâmico no ambiente virtual e tridimensional. Enfim, por meio dos mecanismos de interação, o usuário pode navegar e explorar informações sobre a execução do software, utilizando dispositivos convencionais e não-convencionais. Detalhes sobre estes elementos são descritos nas seções a seguir.

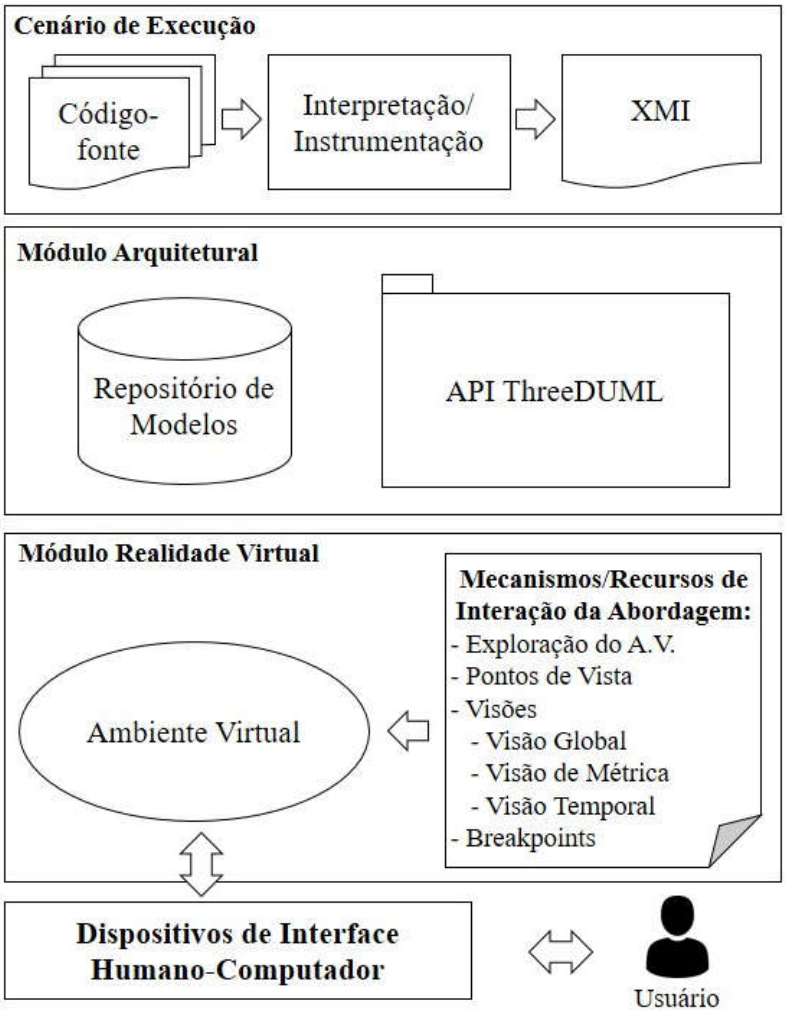


Figura 4.1 – Visão geral da abordagem VisAr3D-Dynamic

4.3.1. Cenário de Execução

A abordagem considera que o comportamento dinâmico do software esteja persistindo no formato XMI (XML *Metadata Interchange*). O XMI é um padrão da OMG (*Object Management Group*) que usa informações UML para criar documentos

XML. Através do uso de XML, informações podem ser transferidas entre os programas de modelagem que tem por base a UML (OMG, 2016). A UML (*Unified Modeling Language*) é uma notação gráfica padrão, proposta pela OMG, que ajuda na descrição e no projeto de sistemas de software orientados a objetos (BOOCH *et al.*, 2005).

Adotando o método de instrumentação ou interpretação, as ferramentas capturam os dados do trechos de código selecionados. Por exemplo, a ferramenta Enterprise Architect (EA) captura a execução do código por meio de *breakpoints* e converte-os automaticamente em diagramas UML. De forma que haja interoperabilidade dos diagramas entre diferentes ferramentas CASE (*Computer-Aided Software Engineering*), estes podem ser exportados como arquivos seguindo o padrão XMI.

4.3.2. Módulo Arquitetural

Estes arquivos XMI, por sua vez, são armazenados no repositório de modelos da abordagem de forma que componham os cenários de execução do software. Cada cenário de execução retrata um escopo de execução com o intuito de observar a execução de acordo com alguns estímulos estabelecidos. Por exemplo, o envio de alguns campos preenchidos de um formulário é um cenário de execução diferente do envio de todos os campos preenchidos do mesmo formulário. Os cenários de execução podem utilizar os mesmos trechos de códigos, porém com entradas diferentes e, consequentemente, podendo também produzir resultados diferentes. No entanto, o XMI como formato interoperável de diagramas UML possui algumas restrições no contexto da análise do comportamento dinâmico de software.

Considerando que cada arquivo XMI corresponde a um cenário de execução, e que alguns cenários podem representar a execução de algumas instâncias de classes em comum, não é possível reconhecer que estas instâncias de cada cenário de execução são as mesmas. Isto ocorre devido a forma como os dados do comportamento dinâmico são persistidos. Na maioria dos casos, esta compreensão da relação entre os elementos de software em diferentes cenários de execução são realizadas de maneira visual pelos engenheiros de software. Adicionalmente, esta análise visual também pode acontecer em cada cenário de execução. Por exemplo, um engenheiro de software pode reconhecer visualmente que uma classe *ClasseA* do diagrama de classes é a mesma *ClasseA* que

está no diagrama de sequência. Contudo, estas classes são tratadas como distintas no XMI. A Figura 4.2 apresenta um exemplo deste tipo de estruturação.

```
<element xmi:idref="EAID_B73C1711_2DBC_4539_86C3_ECE054DCD94E" xmi:type="uml:Class"
name="ClasseA" scope="public">
<element xmi:idref="EAID_720EF248_0DE0_44fa_9B79_E308E41C2139" xmi:type="uml:Sequence"
name="ClasseA " scope="public">
```

Figura 4.2 – Principais elementos dos diagramas no formato XMI

A *tag element* juntamente com seus atributos representa cada elemento do diagrama UML. O atributo *xmi:type* corresponde ao nome do elemento e o atributo *xmi:idref* a identificação deste elemento no arquivo. Estas identificações são utilizadas para referenciar a qual diagrama os elementos fazem parte, conforme demostram a Figura 4.3 e Figura 4.4. O valor do atributo *subject* da *tag element*, a qual pertence hierarquicamente à *tag diagram*, é igual ao *xmi:idref*. Neste caso, a classe *ClasseA* com *xmi:idref = EAID_B73C1711_2DBC_4539_86C3_ECE054DCD9E* pertence ao diagrama de classes (Figura 4.3) e a *lifeline ClasseA* com *xmi:idref = EAID_720EF248_0DE0_44fa_9B79_E308E41C2139* pertence ao diagrama de sequência (Figura 4.4).

```
<diagram xmi:id="EAID_65ED9BA5_6DC3_450a_B7B1_B5A5C2AE1934">
  <properties name="Composite" type="Logical"/>
  <elements>
    <element geometry="Left=892;Top=50;Right=1086;Bottom=502;" subject="
EAID_B73C1711_2DBC_4539_86C3_ECE054DCD94E" seqno="9" style=" DUID=32896732;"/>
</elements>
</diagram>
```

Figura 4.3 – Diagrama de classes no formato XMI

```
<diagram xmi:id="EAID_D0D1DFED_3338_40f8_9CFD_B84C1B70CC73">
  <properties name="Composite_Program_Main" type="Sequence"/>
  <elements>
    <element geometry="Left=474;Top=53;Right=588;Bottom=123;" subject="
EAID_720EF248_0DE0_44fa_9B79_E308E41C2139" seqno="3" style="DUID=8640A331;"/>
  </elements>
</diagram>
```

Figura 4.4 – Diagrama de sequência no formato XMI

4.3.2.1. API ThreeDUML

API (Application Programming Interface – Interface de Programação da Aplicação) é um conjunto de regras e especificações que um software faz uso de serviços e recursos providos por outro software onde está implementada a API. Uma API funciona como a interface entre sistemas de software a fim de facilitar a sua interação, de modo similar à interface entre computadores e pessoas (MURUGESAN *et al.*, 2001). A principal característica da API, do ponto de vista do seu desenvolvimento,

é deixar a complexidade para quem a desenvolveu e apenas conceber aos seus usuários, funções que sejam de fácil acesso para o desenvolvimento de suas próprias aplicações.

A abordagem proposta reconhece cada elemento de software e seus relacionamentos, independente de qual digrama pertence por meio de uma API desenvolvida, denominada ThreeDUML. O objetivo da API é realizar um mapeamento dos diagramas UML persistidos no padrão XMI e apoiar o desenvolvimento de aplicações em RV.

O *design* da API foi inspirado no metamodelo da UML e projetado visando as melhores práticas de reutilização de software para apoiar tanto no seu desenvolvimento quanto em futuras evoluções. A Figura 4.5 apresenta as principais classes da API. Seguindo o padrão dos diagramas UML, cada diagrama possui o objetivo de documentar uma visão específica do software. Por exemplo, ao passo que o diagrama de classes mostra uma visão geral e estrutural do software, o diagrama de sequência representa a visão comportamental, documentando as interações entre objetos por meio da troca de mensagens entre eles.

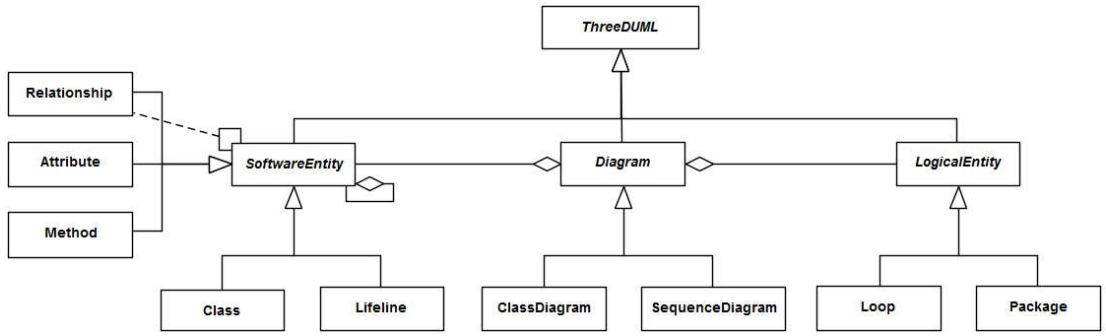


Figura 4.5 – Principais classes da API ThreeDUML

Na API, considera-se pacote, classe, atributo, mensagem e objeto como entidade de software, pois cada uma representa uma particularidade do software. Por exemplo, o diagrama de classes mostra informações sobre o relacionamento entre diferentes classes a atuação de polimorfismo, e o diagrama de sequência documenta os objetos e mensagens trocadas entre eles na ordem de execução. Em alguns casos, é necessário obter uma visão geral de um conjunto de objetos. Cada objeto é uma instância que herda atributos e métodos de sua classe. Neste sentido, uma classe representa um conjunto de objetos. Na análise dinâmica, é comum utilizar diagramas de sequência para documentar os objetos e mensagens trocadas entre eles na ordem de execução. Portanto,

a mesma entidade de software pode ser representada em diferentes contextos, tanto como classe (diagrama de classe) quanto objeto (diagrama de sequência).

Na API, os principais elementos de um diagrama (*e.g.*, classes do diagrama de classes e *lifeline* do diagrama de sequência) são denominados como *SoftwareEntity*, os quais compõem um determinado tipo de diagrama (*e.g.*, *ClassDiagram* e *SequenceDiagram*). A classe *LogicalEntity* corresponde aos elementos visuais complementares dos diagramas UML. Por exemplo, informações sobre a região de *loop* no diagrama de sequência representado por retângulo agrupando alguns métodos, ou então, a informação de multiplicidade nas classes.

4.3.3. Módulo Realidade Virtual

Após o reconhecimento e mapeamento dos diagramas por meio da API, o ambiente virtual é renderizado e os dados do cenário de execução são organizados baseados no conceito de perspectivas de software, a qual é apresentada na seção seguinte.

4.3.3.1. Perspectivas do Software

As notações gráficas de projeto existem há algum tempo e seu principal valor está na comunicação e no entendimento. Um diagrama eficaz frequentemente pode ajudar a transmitir ideias sobre um projeto. Os diagramas também podem ajudar a entender um sistema de software ou um processo de negócio. Como parte de uma equipe tentando descobrir algo, os diagramas ajudam toda a equipe tanto a entender como comunicar esse entendimento. Dessas notações gráficas, a importância da UML é proveniente de seu uso amplo e da padronização dentro da comunidade de desenvolvimento orientado a objetos. A UML se tornou não somente a notação gráfica dominante dentro do mundo orientado a objetos, como também uma técnica popular nos círculos não-orientados a objetos (FOWLER, 2014).

De acordo com FOWLER (2014), a grande motivação de utilizar linguagens gráficas é que as linguagens de programação não estão em um alto nível de abstração suficiente para facilitar as discussões de projeto. Segundo SOMMERVILLE (2011), a UML é utilizada para construir modelos abstratos de um sistema, em que cada modelo apresenta uma visão ou perspectiva, diferente do sistema. A partir de perspectivas diferentes, pode-se desenvolver diversos modelos para representar o sistema, tais como:

- perspectiva externa, para modelar o contexto ou o ambiente do sistema;
- perspectiva de interação, para modelar as interações entre um sistema e seu ambiente ou entre os componentes de um sistema;
- perspectiva estrutural, para modelar a organização de um sistema ou a estrutura dos dados processados pelo sistema; e
- perspectiva comportamental, para modelar o comportamento dinâmico do sistema e como ele reage aos eventos.

Além de ser bastante utilizada na análise e *design*, a UML também transmite informações detalhadas sobre o código em documentos em papel ou via um navegador interativo em projetos de engenharia reversa. Os projetos podem mostrar, de forma gráfica, cada detalhe sobre uma classe, que é mais fácil para os desenvolvedores entenderem (FOWLER, 2014).

No contexto de análise dinâmica, os mantenedores precisam explorar informações do software que vão além de rotas de execução. Além de compreender a troca de mensagens, o código-fonte, as classes que instanciaram os objetos, bem como os pacotes que armazenam e organizam as classes, podem ser consultados a fim de apoiar na compreensão da execução de modo geral. Desta forma, estas informações extras são tratadas como perspectivas do software e cada uma é representada por um diagrama UML e o código-fonte. As formas como as perspectivas do software são organizadas no ambiente virtual e tridimensional seguem uma ordem estabelecida ao longo do eixo *z* (Figura 4.6):

- -1: Código-fonte;
- 0: Diagrama de sequência;
- 1: Digrama de classes;
- 2: Diagrama de pacotes.

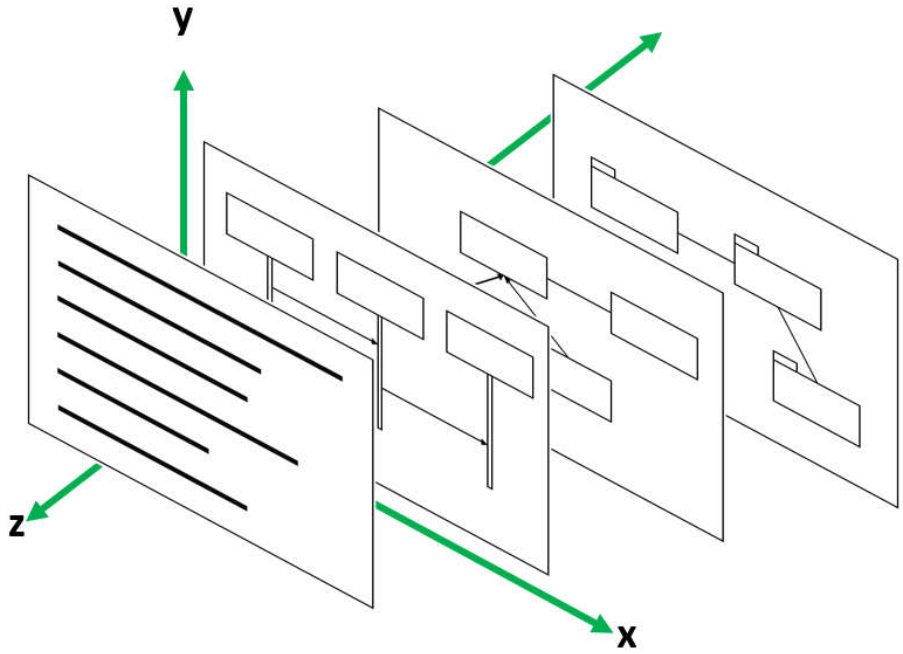


Figura 4.6 – Organização das perspectivas do software no ambiente virtual

O principal diagrama no ambiente virtual é o de sequência. Por ser a única perspectiva que descreve o comportamento do software, seu valor no eixo *z* é igual a 0. A partir desta, as outras perspectivas com valor positivo são organizadas de acordo com o nível de abstração. Neste sentido, a próxima perspectiva é o diagrama de classes e, posteriormente o diagrama de pacotes com os valores 1 e 2 no eixo *z*, respectivamente. Portanto, conforme o valor de *z* se distancia de 0, maior é a abstração do software. O código-fonte é considerado a perspectiva de baixo nível de abstração. Deste modo, seu valor no eixo *z* recebe -1.

4.3.4. Dispositivos de Interação Humano-Computador

Dispositivos convencionais, tais como teclado e mouse, podem ser utilizados pelos participantes como forma de interação com AVs. No entanto, dispositivos não convencionais possuem a característica de facilitar a interação com objetos virtuais e tridimensionais, bem como potencializar a sensação de imersão utilizando até os cinco sentidos humanos simultaneamente.

Estas interfaces podem ser classificadas em dispositivos de entrada e de saída. Os de entrada são dispositivos que os participantes utilizam para enviar algum sinal para o ambiente virtual, por exemplo, o teclado. E os dispositivos de saída são aqueles que o ambiente virtual envia alguma informação para o participante, como o monitor, por exemplo.

A proposta da abordagem vai além de criar diagramas UML ou metáforas em 3D. O diferencial está na forma como interagir com o software, preservando o volume de dados necessários para explorar informações e apoiar a compreensão do comportamento dinâmico de sistemas de software por meio de RV. A partir do ambiente virtual construído, alguns recursos podem ser acessados, os quais serão apresentados na próxima seção.

4.4. Recursos de Interação da Abordagem

Esta seção apresenta os recursos da abordagem proposta nesta dissertação, os quais foram implementados no protótipo. A maioria destes recursos são reutilizados e adaptados da abordagem VisAr3D (RODRIGUES, 2012). No entanto, os recursos Visão Temporal, Menu de Contexto e *Breakpoints* são contribuições deste trabalho, dado o foco em análise dinâmica e na organização das perspectivas do software, bem como o desenvolvimento da API ThreeDUML.

4.4.1. Visão Global

Todas as perspectivas do software são visualizadas seguindo uma organização do espaço 3D apresentada anteriormente. Todos os diagramas mostram seus elementos básicos para o entendimento comum por parte dos mantenedores. Em relação ao código-fonte, somente os trechos de códigos que foram executados são exibidos (RODRIGUES, 2012).

Nesta visão global, todos os recursos da abordagem podem ser acessados, principalmente o acesso às informações, mediante a interação com o *mouse*, tais como o acesso ao Menu de Contexto, Visões e Pontos de Vista.

4.4.2. Exploração do Ambiente 3D

O ambiente virtual é exibido em uma janela que possui botões de navegação e interação na sua parte inferior. Através destes botões, o mantenedor, como se estivesse imerso no ambiente virtual, pode explorar o ambiente com a ajuda do mouse, movendo-se no espaço para a direita, para a esquerda, para cima, para baixo, para longe ou para próximo da perspectiva, com ângulos de rotação. A RV como interface avançada para aplicações computacionais pode ainda fazer uso de dispositivos multisensoriais, que capturam seus movimentos e comportamento e reagem a eles, como luvas, capacetes, caves, entre outros, por exemplo (RODRIGUES, 2012).

4.4.3. Pontos de Vista

Este recurso é usado para definir as posições para a visualização do mundo virtual e proporcionam visitas guiadas no formato de animação. Por exemplo, a partir de um objeto no diagrama de sequência, pode ser acessado a classe correspondente a este objeto no digrama de classes. O ponto de vista facilita a navegação do usuário através de pontos de interesse pré-definidos. Este recurso ajuda a preservar o usuário no seu "mapa mental", possibilitando a visão de todos os elementos relacionados em um mesmo campo visual e facilitando a evocação de conhecimento. Métodos rápidos de navegação são essenciais para que o usuário aprenda como as partes se relacionam entre si. Sem isso, torna-se muito difícil para o usuário desenvolver um mapa mental do modelo (FRANCK *et al.,* 1995).

4.4.4. Visões

A VisAr3D-Dynamic apresenta um conjunto de visões que permitem diferentes formas de visualização. Ela possibilita a navegação e visualização dos diagramas do sistema, no qual diferentes aspectos ou perspectivas do sistema podem ser focados independentemente. A abordagem permite aos usuários explorarem as diversas visões, sem forçá-los a seguir um caminho específico.

4.4.4.1. Visão de Métricas

A principal vantagem do uso de métricas dinâmicas na engenharia de software é a capacidade de medir com mais precisão os atributos internos do software (*e.g.*, acoplamento, complexidade) que possuem impacto direto sobre fatores de qualidade de software, tais como confiabilidade, testabilidade, reutilização, manutenção, desempenho, dentre outros. Na literatura técnica foram propostas diversas métricas dinâmicas, cada uma medindo uma característica do software. Dentre elas, a métrica de acoplamento dinâmico é usada para medir o acoplamento real que ocorre entre um par de objetos ou classes em tempo de execução em um sistema de software (CHHABRA & GUPTA, 2010).

No entanto, à medida que a aplicação de métricas pode resultar em uma enorme quantidade de dados, e esses dados são normalmente apresentados em tabelas, é um trabalho difícil para um engenheiro de software fazer o mapeamento entre os valores das métricas de uma tabela e os objetos do diagrama UML, manualmente. A abordagem combina o *layout* existente dos diagramas com a visualização da métrica de

acoplamento dinâmico EOC (*Export Object Coupling*), cujo valor pode ser encontrado através da equação:

$$EOC_x(o_i,o_j)=\frac{\left|\{M_x(o_i,o_j)|o_i,o_j \in O \wedge o_i \neq o_j\}\right|}{MT_x} \times 100$$

onde: $M_x(o_i, o_j)$ é o número de mensagens enviadas de o_i para o_j e MT_x é o total de mensagens trocadas durante a execução do cenário x (YACOUB *et al.*, 2000).

4.4.4.2. Visão dos Relacionamentos com Outros Tipos de Diagramas

Considerando que as entidades de software podem ser representadas em diferentes tipos de diagramas e em contextos diferentes, esta visão tem como objetivo explicitar o relacionamento de cada elemento através dos diagramas disponíveis no ambiente virtual. Este tipo de relacionamento pode ser percebido através de linhas que interligam os elementos que compõem cada perspectiva no ambiente virtual. Desta maneira, pode-se perceber o relacionamento do elemento nas diferentes perspectivas do software em uma única visão.

4.4.4.3. Visão Temporal

Esse recurso da abordagem é um dos diferenciais em relação às outras técnicas de análise dinâmica apresentadas na Seção 3.4. Em se tratando de análise dinâmica, grande parte das abordagens constroem as visualizações do comportamento dinâmico de forma estática. Neste sentido, recursos visuais interativos podem aprimorar na obtenção de *insights* de sistemas complexos (MALETIC *et al.*, 2001).

Um dos potenciais da RV é a simulação de eventos reais através do ambiente virtual disponível (KIRNER & TORI, 2004). Esta propriedade é utilizada na abordagem como forma de controle da execução do software, análogo a um controle de vídeo. O objetivo é proporcionar um controle interativo que forneça a visualização da execução de cada elemento de software nas diferentes perspectivas compostas no ambiente virtual. Através da barra de navegação, o usuário pode controlar o sentido da execução (avançar ou recuar), percebendo a execução de cada elemento nas diferentes perspectivas.

Quando a execução é iniciada, o objeto e sua mensagem aparecem no diagrama de sequência e, em paralelo, a classe correspondente a esse objeto, também aparece no diagrama de classe, bem como o pacote da classe.

4.4.5. Breakpoints

Teste e depuração são duas atividades que têm um impacto importante no curso e na qualidade do produto gerado durante o processo de software. Teste de software é o processo de execução de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado. Após a execução dos testes, é necessária a execução de um processo de depuração para a identificação e correção dos defeitos que originaram falhas (CRAIG & JASKIEL, 2002).

Breakpoint é um dos recursos mais conhecidos no processo de depuração e é muito útil no desenvolvimento, pois permite descobrir o que acontece a cada linha de código em tempo de execução. Os *breakpoints* no ambiente virtual da abordagem funcionam com indicadores visuais de execução dos métodos. Ao acessar o código-fonte de uma determinada classe, somente os métodos que foram executados serão visualizados e à frente deles estarão os *breakpoints* informando a ordem em que foram executados, de acordo com o diagrama de sequência.

4.5. Considerações Finais

Este capítulo apresentou a abordagem proposta que utiliza Realidade Virtual como uma nova técnica para apoiar a compreensão do comportamento dinâmico de sistemas de larga escala. A partir dos dados gerados da execução, a API ThreeDUML, pertencente ao Módulo Arquitetural, possui a responsabilidade de ler os arquivos XMI e reconhecer cada elemento de software e seus relacionamentos, de forma que sejam visualizados no ambiente virtual, após o processamento das informações de execução no Módulo Realidade Virtual. Com o ambiente concebido, os usuários podem explorar o software através dos recursos providos pela abordagem, tais como visões, pontos de vista e *breakpoints*.

Em relação aos requisitos, a Tabela 4.1 mostra o comparativo entre os trabalhos relacionados com representação 3D e a abordagem VisAr3D-Dynamic. A abordagem proposta nesta dissertação possui o diferencial de permitir a visualização do código-fonte e os vínculos de cada elemento de software nas diferentes perspectivas, além de explicitar o fluxo de mensagens entre objetos por meio de animações.

Os requisitos que não pertenciam ao escopo desta pesquisa não foram atendidos. Por exemplo, entende-se que o problema de reduzir informação do comportamento dinâmico requer mais esforços, visto que já existem vários trabalhos na literatura

técnica que tratam sobre este problema. Futuramente, pode-se agregar técnicas existentes ou propor novas maneiras de reduzir informação. Sendo um ambiente virtual tridimensional e ilimitado, não faz sentido continuar utilizando diferentes visões. Portanto, foi implementada uma visão, mas unificando diferentes perspectivas do software e mantendo visualmente o relacionamento entre elas.

Tabela 4.1 – Comparativo entre os trabalhos relacionados em 3D e a abordagem VisAr3D-Dynamic

Requisitos		**ABORDAGENS**					
		EVOSPACES	**3D-PP**	**EXPLORVIZ**	**GREEVY**	**ZHAO**	**VisAr3D-Dynamic**
Tipo do dado	Estrutural	+	-	+	+	-	+
	Comportamental	+	+	+	+	+	+
Abstração	Redução de informação	-	-		-	+	-
Visão	Uma	+	+	-	+	+	+
	Múltiplas	-	-	+	-	-	-
Qualidade de software	Métricas	+	-	+	-	-	+
Código-fonte	Edição	-	-	-	-	-	-
	Visualização	-	-	-	-	-	+
Dispositivo	Convencionais	+	+	+	+	-	+
	Não-convencionais	-	-	-	-	+	-
Representação Gráfica	UML	-	-	-	-	+/-	+
	Polimétrica	-	+	+	+	-	-
	Metáfora	+	-	+	-	-	-
Interatividade	Recursos interativos	+	+	-	+	-	+

+ Requisito atendido - Requisito não atendido +/- Requisito parcialmente atendido

Nenhuma abordagem possibilita a edição e, portanto, seria um grande diferencial deste trabalho. Contudo, entende-se que a forma como a visualização do software é transmitida ao usuário, possibilitar a edição poderia ser um fator de risco e aumento na sobrecarga cognitiva ao invés de ajudar nas tarefas de compreensão. Apesar de uma abordagem ter utilizado dispositivo não-convencional, não gerou muito impacto positivo, visto que o rastreador ocular não é muito intuitivo e apresenta atraso no tempo de reposta durante a interação. Utilizar óculos de RV e sensores de gestos, demandam estudos mais concentrados sobre imersão ao ambiente virtual e responder perguntas do tipo, como transmitir a sensação de estar dentro do software, visto que software é intangível e invisível? Por fim, em relação ao tipo de representação gráfica, optou-se

por utilizar a notação UML por ser amplamente utilizada tanto na academia quanto na indústria.

A fim de aplicar a abordagem em situações comuns e verificar vantagens e limitações, um protótipo permite uma análise mais apropriada. A implementação da abordagem em uma plataforma é apresentada no capítulo a seguir.

CAPÍTULO 5 - IMPLEMENTAÇÃO

5.1. Introdução

Com a abordagem apresentada e detalhada anteriormente, é importante descrever e desenvolver meios para sua real adoção. Nesse sentido, a implementação da abordagem permite verificar sua viabilidade, bem como a constatação da necessidade de novos requisitos ou de limitações não previstas. Com um protótipo desenvolvido, pode-se seguir todos os passos de utilização permitidos, além de analisar se sua adoção poderia ocorrer de forma intuitiva.

Além desta seção introdutória, este capítulo apresenta os requisitos de implementação do protótipo na Seção 5.2. Um estudo sobre a viabilidade de tecnologias tridimensionais é apresentado na Seção 5.3. Na Seção 5.4 é exposto a visão geral, as funcionalidades, bem como um exemplo de uso do protótipo. Finalmente, as considerações finais são feitas na Seção 5.5.

5.2. Requisitos

Após a análise dos trabalhos relacionados encontrados por meio da revisão *ad-hoc* da literatura e considerados os pontos positivos e negativos da avaliação heurística da abordagem com requisitos parcialmente implementados (FERNANDES *et al.*, 2016), foram estabelecidos alguns requisitos para a implantação e desenvolvimento do protótipo da abordagem VisAr3D-Dynamic, a fim de viabilizar o estudo de observação. Os requisitos são apresentados a seguir.

- **Importação dos dados:** é comum em ferramentas de análise dinâmica obter os dados da execução do software por meio de instrumentação ou interpretação. Por exemplo, ZHAO *et al.* (2009) coletam rotas de execução por meio de instrumentação via programação orientada a aspecto, devido a vantagem de compilação em conjunto com os códigos em Java dos programas de origem. No entanto, como a abordagem proposta nesta dissertação se baseia na VisAr3D, o protótipo deve ter a capacidade de ler dos dados da execução do software no formato XMI e gerar as visualizações tridimensionais automaticamente. Além disso, em virtude da potencial utilização de diagramas UML em ambientes virtuais e tridimensionais, tanto pelo grupo de pesquisa quanto pela comunidade de desenvolvimento, foi identificada uma oportunidade de desenvolver uma API

para apoiar a construção de aplicações interativas que necessitam da renderização de diagramas UML.

- **Relacionamento entre as perspectivas de software no ambiente virtual:** como a ideia da abordagem é facilitar a compreensão do comportamento dinâmico por permitir a exploração de informação de software através da navegação em diferentes perspectivas, o protótipo deve explicitar visualmente o relacionamento de cada elemento com outras perspectivas. Nos diagramas UML é comum expressar o relacionamento entre entidades por meio de uma linha, como por exemplo nos casos de generalização e especialização. Nos trabalhos EXPLORVIZ, GREEVY *et al*. (2006) e ZHAO *et al.* (2009) também utilizam linhas para indicar relação entre as entidades. Portanto, o ambiente virtual deve ser capaz de exibir o relacionamento de cada elemento à outra perspectiva. Por exemplo, criar uma linha entre uma *lifeline*, a qual está na perspectiva dinâmica, e sua classe, a qual está na perspectiva de classes.

- **Visualização da execução passo a passo:** animação ou representação visual dinâmica, por sua natureza, consiste em uma série de informações visuais apresentadas sequencialmente. MAYER & CHANDLER (2001) afirmam que animações podem ajudar na compreensão quando informações são segmentadas e permitem controle da visualização dinâmica. EVOSPACES, 3D-PP e GREVYY *et al.* (2006) utilizam da animação como um recurso interativo para apoiar no entendimento de software complexo. Neste sentido, o ambiente virtual fornecerá meios para que a visualização da execução seja percebida passo a passo, além de permitir o controle sobre a animação.

- **Visualização de todos os elementos:** em algumas aplicações, é comum fazer uso de princípios e técnicas de visualização para permitir a interação e a manipulação por parte do usuário (SHNEIDERMAN, 1996). Dentre elas, a técnica de visão geral será adotada a fim de permitir que, além de o usuário ter a possibilidade de visualizar cada elemento passo a passo nas diferentes perspectivas do software, o ambiente permitirá a visualização de todos os elementos simultaneamente;

- **Visualização de métricas:** as métricas são úteis por inferirem aspectos de qualidade do software. É comum em visualizações tridimensionais se fazer uso

de métricas para ajudar na construção das formas em 3D, como em EVOSPACES e EXPLORVIZ. Neste sentido, o protótipo implementará a métrica de acoplamento dinâmico alterando a forma geométrica em *z*, ou seja, seu comprimento, de cada *lifeline* do digrama de sequência.

- **Interação com o ambiente virtual:** como todo sistema de visualização de software, é fornecido meio para interação com as representações gráficas. No caso do ambiente virtual da abordagem, mais interações poderão ser exploradas devido aos seis graus de liberdade[6]. Neste sentido, interações com o teclado e funções diferenciadas explorando os três botões do *mouse* serão fornecidas.

5.3. Viabilidade Tecnológica

Antes de iniciar a implementação do protótipo final, foi realizada uma verificação quanto a viabilidade de tecnologias 3D. Primeiramente, foi decidido implementar um protótipo que animasse o diagrama de sequência, pois este é um tipo de diagrama comportamental, ao qual as técnicas de animação poderiam ser aplicadas. O protótipo da VisAr3D foi implementado com a linguagem Xj3D[7] para construir os modelos virtuais. Contudo, em julho de 2014, constatou-se que esta linguagem foi descontinuada. A falta de suporte e atualizações poderiam comprometer o desenvolvimento do protótipo final pretendido. Em virtude disto, optou-se por desenvolver em Java 3D, por se assemelhar com o Xj3D.

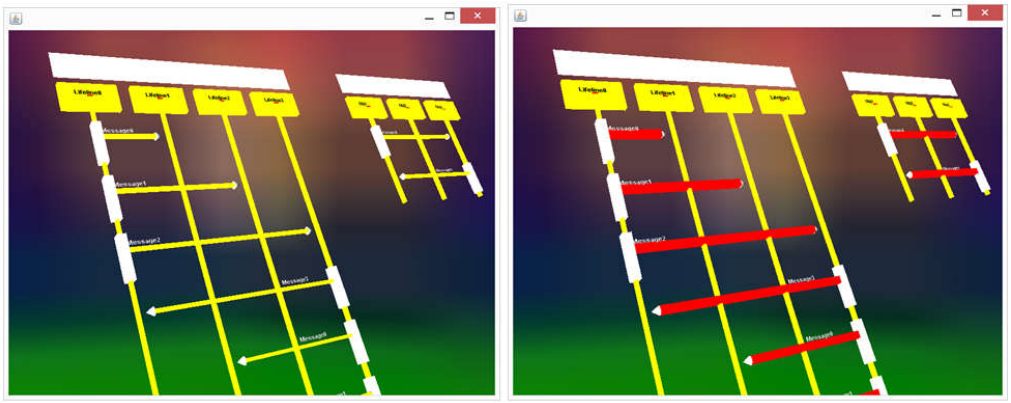
A API Java 3D, desenvolvida pela Sun Microsystems e mantida atualmente pela Java.net, fornece uma coleção de funções em alto nível na linguagem Java para criar, renderizar e manipular um grafo de cena 3D composto por objetos do tipo geométrico, luzes, materiais, sons e outros (DAVISON, 2005). A Figura 5.1 mostra o protótipo inicial desenvolvido.

Na Figura 5.1 (a), modelos de diagramas de sequência são renderizados a partir de arquivos XMI e inseridos no ambiente virtual. Os fluxos de mensagens entre objetos são percebidos por meio de retângulos vermelhos que envolvem as setas. Na Figura 5.1 (b), os mesmos modelos possuem a representação da animação. Para se ter a ideia da

[6] Grau de liberdade é o número de eixos de coordenadas que podem ser manipulados simultaneamente durante o processo interativo

[7] http://www.xj3d.org/

ordem da troca de mensagens, a animação acontece da seguinte maneira: a barra vermelha envolve a seta da mensagem com o efeito *fade-in* e, depois de alguns instantes, a barra desaparece com o efeito *fade-out*. Caso haja uma próxima animação, o início desta, começará junto com o término da animação anterior.



(a) (b)

Figura 5.1 – Protótipo implementado em Java 3D

No entanto, programar em Java 3D não é uma tarefa trivial. Modelar um objeto em 3D, em nível de código, requer domínio da linguagem e alto poder de abstração, pois se modela escrevendo o código e imaginando como ficará em cena. Com recursos de animação, a complexidade aumenta. Por outro lado, construir animações em *engines*, as quais possuem *timelines* e recursos específicos, ações predefinidas e outros suportes facilitam o desenvolvimento. *Engines* de jogos como Unity 3D[8] são mais fáceis de desenvolver, pois fornecem recursos que facilitam a construção dos objetos.

A estrutura do programa, painéis, ícones e principalmente a visualização em tempo real do que está sendo modelado ajuda no desenvolvimento, tornando-o mais ágil e produtivo e minimizando futuros erros. A Unity 3D tem o potencial de facilitar o desenvolvimento de jogos para várias plataformas diferentes. Essa ferramenta permite a utilização de *scripts* tanto na linguagem C# como em JavaScript. Além disso, permite a manipulação de elementos criados nas principais aplicações do gênero, como Maya[9] e Blender[10]. Portanto, dadas essas vantagens da *engine*, o protótipo final foi implementado usando a *engine* Unity 3D com a linguagem de programação C#.

[8] http://unity3d.com/

[9] http://www.autodesk.com.br/products/maya/overview

[10] https://www.blender.org/download/

5.4. O Protótipo VisAr3D-Dynamic

Com as experiências de implementação citadas acima e a possível utilização de outros tipos de diagramas UML em 3D pelo mesmo grupo de pesquisa, ou até por desenvolvedores interessados no desenvolvimento de aplicações interativas em 3D no contexto de engenharia de software, decidiu-se desenvolver uma API para facilitar na conversão de modelos em 2D para o 3D, como foi apresentado na Seção 4.3.2.1. Portanto, o desenvolvimento do protótipo envolveu o uso do Unity 3D com a linguagem de programação C#, juntamente com o desenvolvimento e integração da API ThreeDUML. Nesta seção, é apresentada uma visão geral das funcionalidades do protótipo, bem como um exemplo de uso do protótipo.

5.4.1. O protótipo

Ao iniciar o protótipo são exibidos os componentes do menu inferior e nenhuma perspectiva do software é exibida, como mostra a Figura 5.2. Neste menu, estão as opções de controlar a simulação da execução, bem como a visualização do relacionamento com outros tipos de diagramas, métrica de acoplamento dinâmico e apoio em casos de desorientação espacial no ambiente virtual.

A Figura 5.3 (a) mostra as funções de desorientação espacial, visualização do relacionamento com outros diagramas e métrica. O botão *Camera Reset* tem o objetivo de apoiar a navegação durante a exploração de informações acerca do comportamento dinâmico. Pode ocorrer em alguns usuários uma desorientação espacial no ambiente virtual, ou seja, no momento da exploração não identificar em que lugar está no ambiente. Portanto, ao acionar *Camera Reset,* o usuário é levado automaticamente para a primeira mensagem do digrama de sequência.

Em qualquer momento, pode ser visto o relacionamento de cada elemento através das diferentes perspectivas do software. O botão *Relationship between layers* habilita ou desabilita a visualização do relacionamento com outros tipos de diagramas e do código-fonte. Também em qualquer momento, a métrica de acoplamento dinâmico pode ser visualizado, por meio do *checkbox Dynamic Coupling*.

Na Figura 5.3 (b), estão os controles para manipular a simulação da execução do software em cada perspectiva. O botão *Play* inicia a execução, o botão *Stop* para a execução e volta ao início da execução, *Rewind* volta uma mensagem executada e pausa, *Forward* avança uma mensagem e pausa e, por fim, o *Slider* (barra horizontal

acima dos botões) permite o controle manual da execução tanto para avançar quanto para voltar.

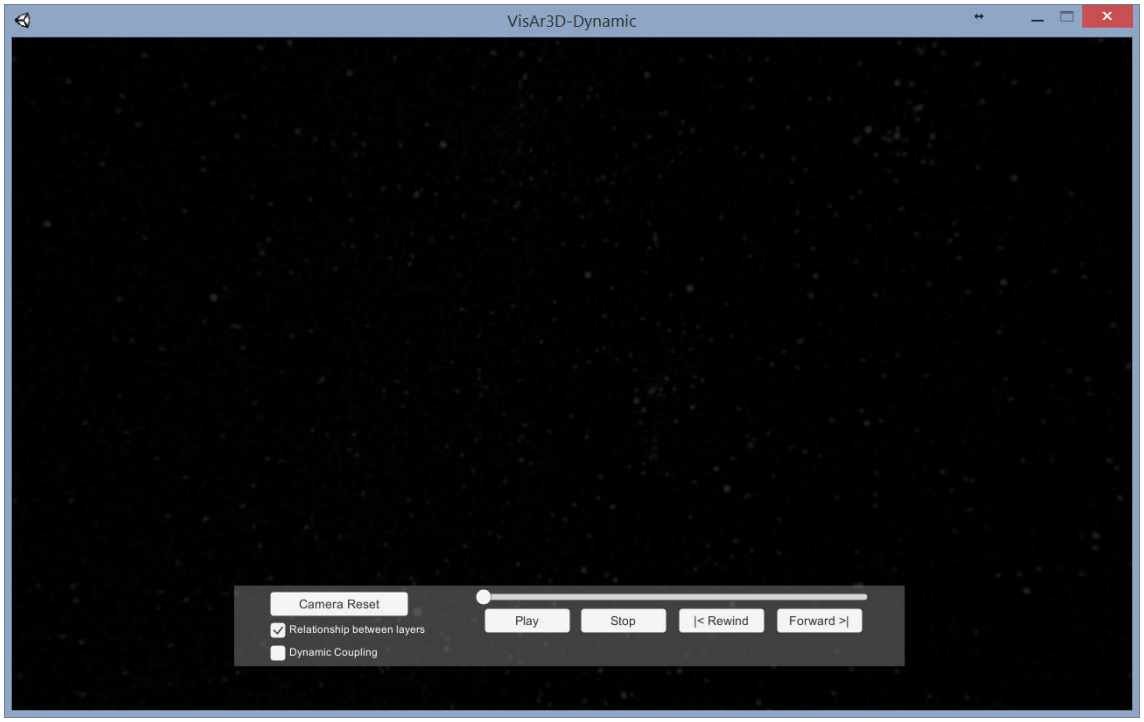


Figura 5.2 – Tela inicial do protótipo

Figura 5.3 – Menu inferior e seus componentes

Ao clicar com o botão direito do *mouse* em cima de qualquer elemento do diagrama, é aberto o Menu de Contexto (Figura 5.4). Este menu contém ações que implementam o recurso de pontos de vista da abordagem, conforme apresentado na Seção Capítulo 4 -. Cada botão leva o usuário automaticamente do elemento de origem para o elemento de destino, de acordo com o botão acionado. Os botões são: *Go to the Package* (vai para o pacote); *Go to the Class* (vai para a classe); *Go to the Lifeline* (vai para a *lifeline*); *Go to the Message in Lifeline* (vai para a mensagem da *lifeline* do diagrama de sequência); *Go to the Code* (vai para o código); *Go to the Message in Code* (vai para a mensagem no código-fonte) e *Go to the Next Breakpoint* (vai para a próxima mensagem executada no código-fonte).

Figura 5.4 – Botões do Menu de Contexto

As interações do usuário com o ambiente virtual são realizadas através de dispositivos tradicionais, tais como o teclado e *mouse*. Por meio do teclado, é possível movimentar-se à esquerda, direita, para cima ou para baixo. Normalmente, em aplicações em 3D, as teclas para esta movimentação são ← ↓ → ↑. Além disso, também foram implementadas as mesmas funções para as teclas A, S, D e W. A Figura 5.5 mostra a relação entre os dois conjuntos de teclas de direcionamento, ou seja, a tecla A corresponde à esquerda, S para baixo, D à direta e W para cima.

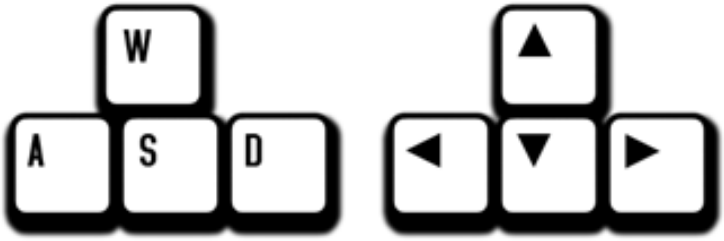


Figura 5.5 – Interações com o teclado

Ao passo que o teclado permite a movimentação em duas dimensões (x e y), o *mouse* permite girar os objetos, proporcionando mais graus de liberdade por meio do botão esquerdo, conforme a Figura 5.6 (a). Em outras palavras, possibilita a visualização e percepção de profundidade dos objetos e do ambiente virtual. Além disso, por meio do *mouse*, é possível aumentar e diminuir o *zoom* utilizando o *scroll*, localizado no centro do mouse, como indicado na Figura 5.6 (b). O botão direito, conforme a Figura 5.6 (c), abre o Menu de Contexto. A Figura 5.7 mostra o resultado da

interação por meio do *mouse* com os botões esquerdo e *scroll* e as teclas de direcionamento do teclado.

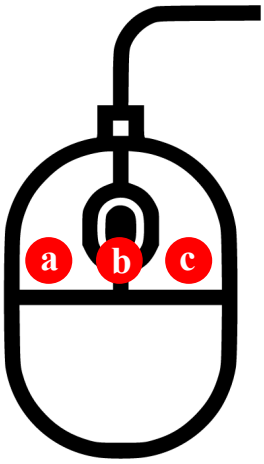


Figura 5.6 – Interações com o mouse

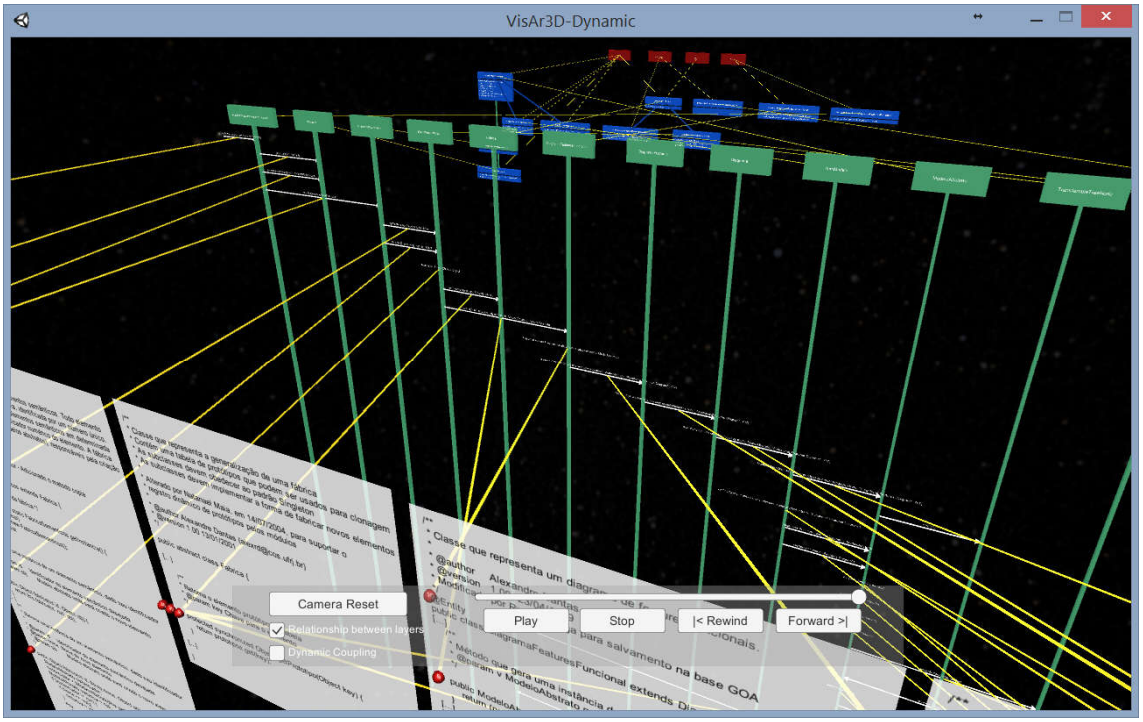


Figura 5.7 – Resultado da interação por meio do teclado e *mouse*

Por fim, os *breakpoints* aparecem à frente de cada mensagem no código-fonte, indicando a ordem de execução de acordo com o diagrama de sequência UML. Nos casos de repetição, é adicionado outro *breakpoint* com o número de execução diferente do anterior.

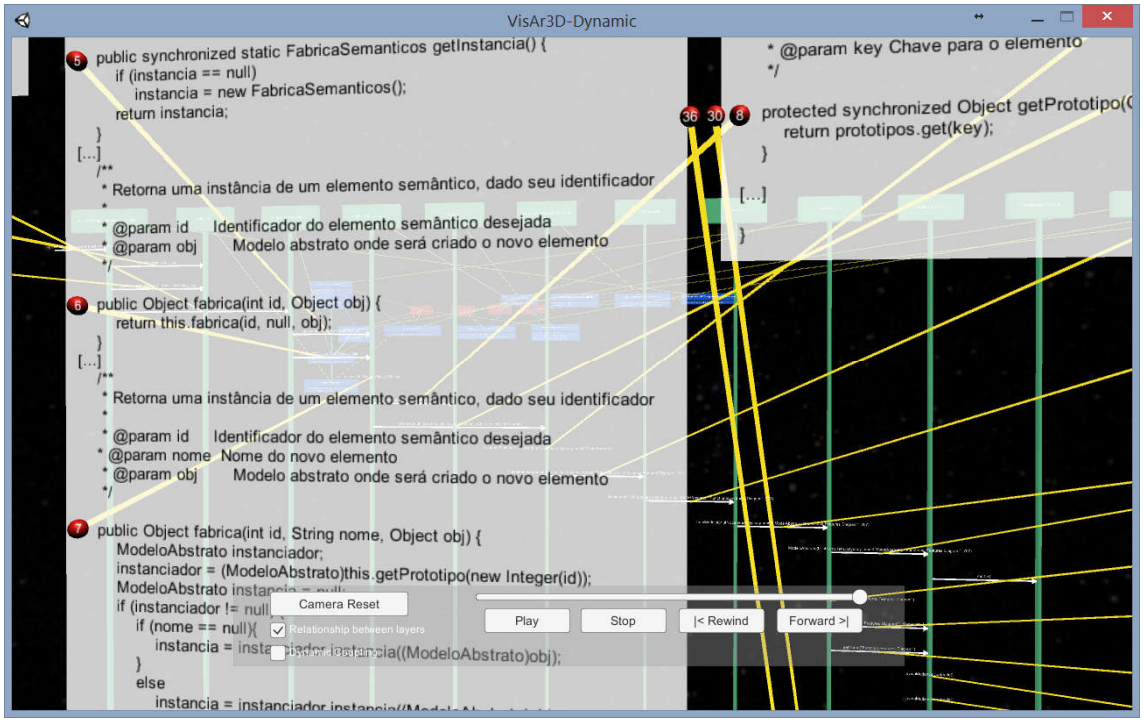


Figura 5.8 – O código-fonte e seus *breakpoints*

5.4.2. Exemplo de Uso

A primeira tela exibe somente o menu inferior, como já apresentado na Figura 5.2. Quando o *Play* é acionado, cada elemento nas diferentes perspectivas é exibido. No entanto, sem a movimentação no ambiente virtual utilizando o teclado e *mouse* dificulta a percepção desta ação (Figura 5.9). Portanto, a movimentação no ambiente é muito importante para explorar informações e perceber a execução de cada elemento nas diferentes perspectivas e seus relacionamentos.

Geralmente, os *players* de vídeos utilizam os segundos como a unidade mínima de execução, ou seja, a cada segundo o vídeo é executado. Além disso, o controle também é similar com as interfaces dos *players* de vídeo. No entanto, a execução no protótipo é orientada à mensagem. Em outras palavras, a cada mensagem executada os elementos relacionados nas outras perspectivas são exibidos. Por exemplo, a Figura 5.10 mostra três mensagens executadas. Neste contexto, o digrama de sequência possui duas *lifelines*, as quais cada uma está relacionada com sua classe no diagrama de classes, que por sua vez está relacionada com seu pacote no diagrama de pacotes. Finalmente, o código das três mensagens é exibido na perspectiva de código-fonte. A Figura 5.10, Figura 5.11 e Figura 5.12 mostram a execução até o fim, no mesmo ângulo, para explicitar a execução mensagem à mensagem nas diferentes perspectivas.

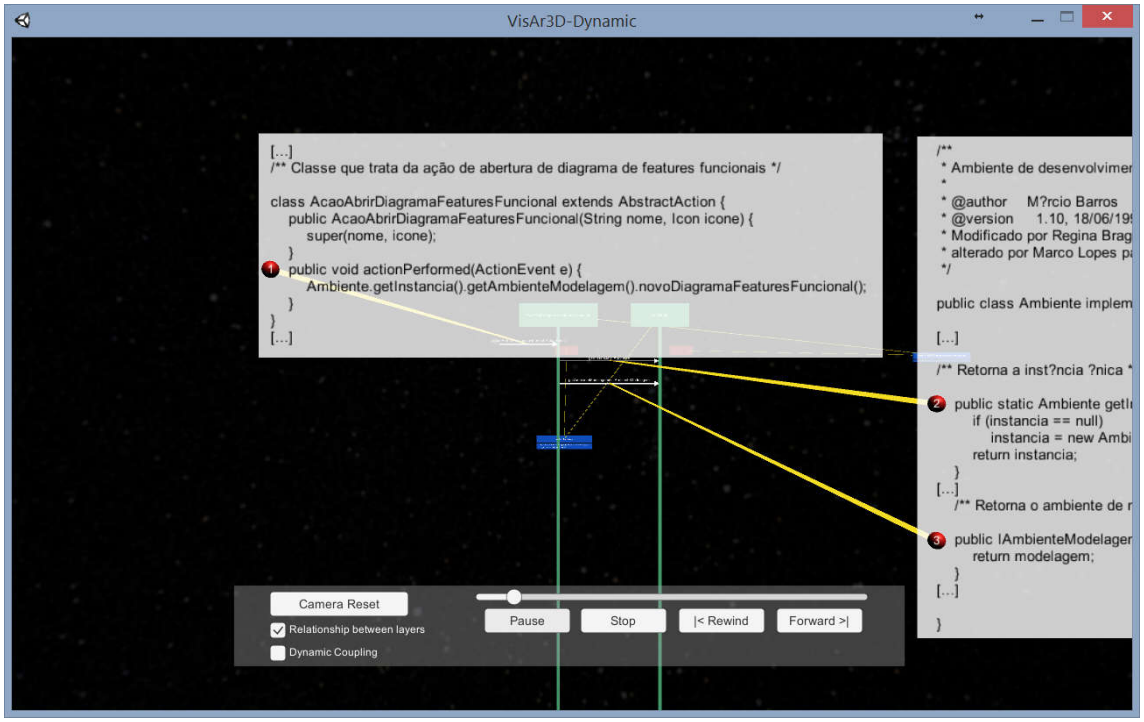


Figura 5.9 – Visão frontal

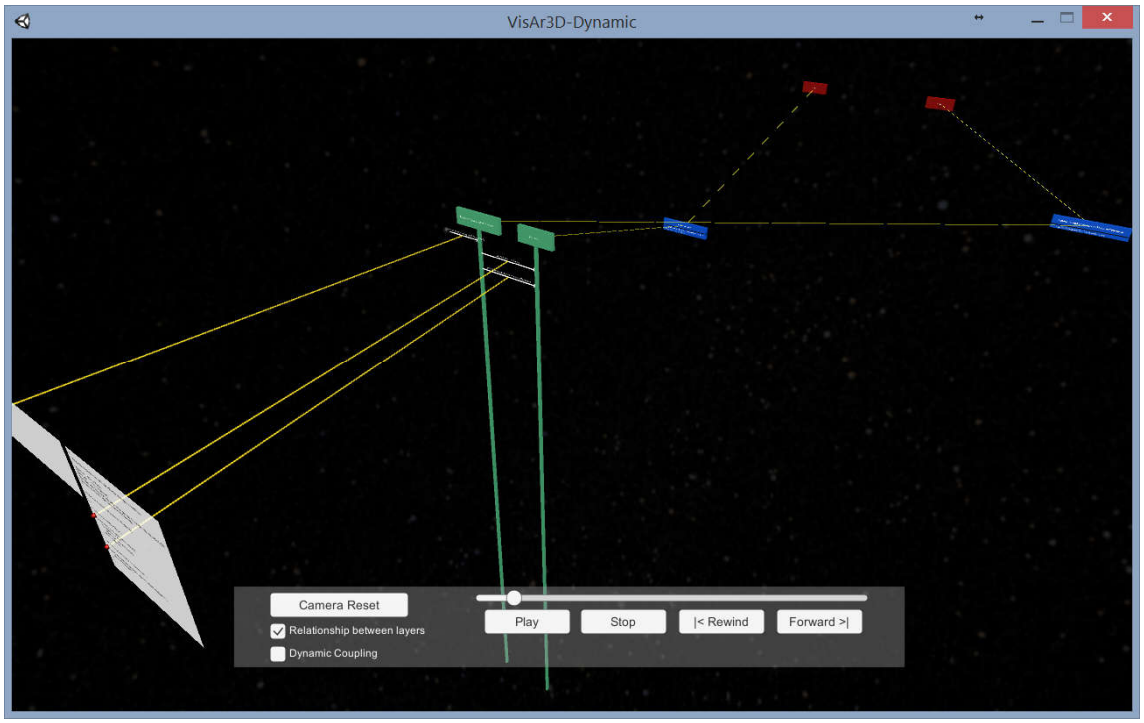


Figura 5.10 – Visualização das perspectivas com três mensagens executadas

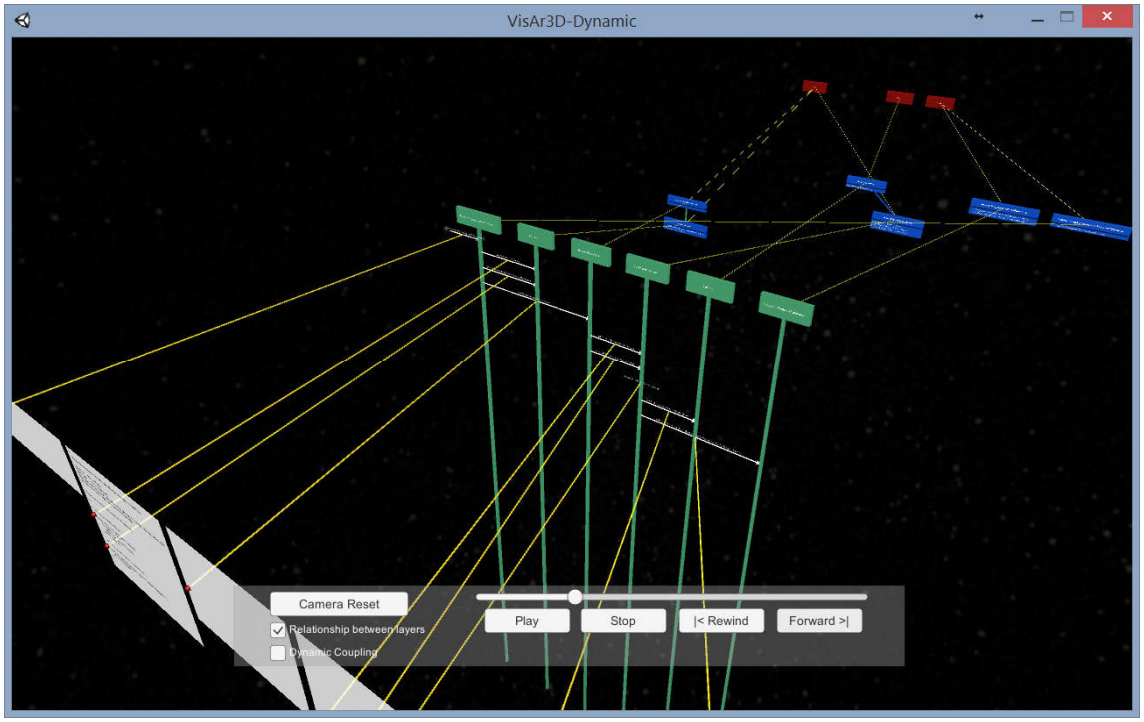


Figura 5.11 – Visualização das perspectivas com execução parcial

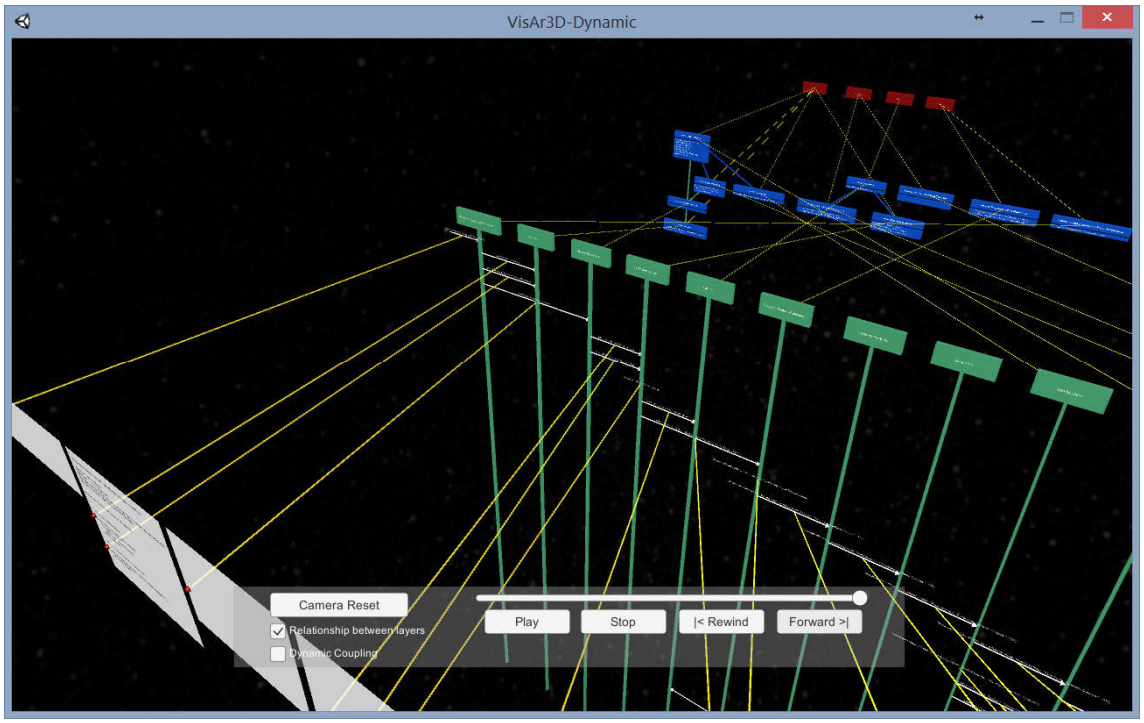


Figura 5.12 – Visualização das perspectivas com execução até o final

Na perspectiva de código-fonte, cada arquivo contém o código das classes que foram executadas no cenário de execução. Porém, com o intuito de não sobrecarregar visualmente o ambiente, somente os comentários, o nome da classe e as mensagens que foram executadas são exibidos. O restante do conteúdo do arquivo é agrupado e

indicado pelos caracteres “[...]”. Além disso, à frente de cada método, são exibidos *breakpoints* que informam a ordem de execução em relação ao diagrama de sequência. Por exemplo, a Figura 5.13 mostra que o método *getPrototipo* da classe *Fabrica* foi executado três vezes e corresponde à oitava, trigésima e trigésima sexta mensagens executadas no diagrama de sequência. Um vídeo[11] de demonstração do protótipo está disponível para ajudar no entendimento da interação com o protótipo.

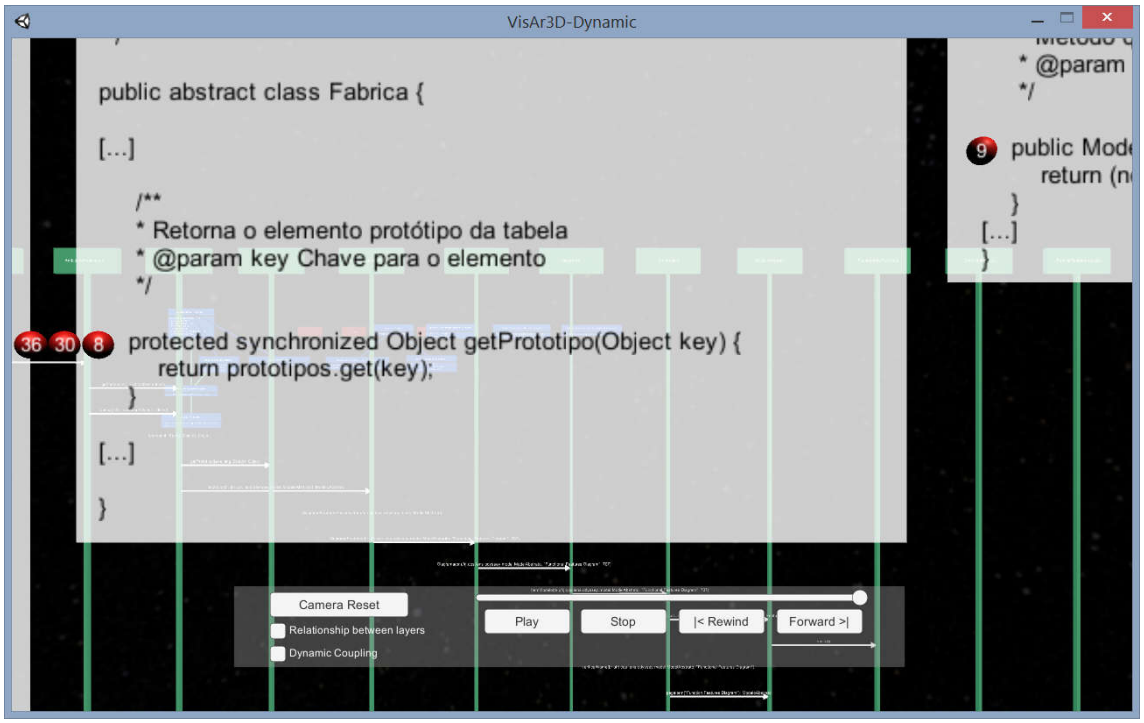


Figura 5.13 – Visualização dos *breakpoints*

5.5. Considerações Finais

Este capítulo detalhou o desenvolvimento do protótipo da abordagem VisAr3D-Dynamic. Adicionalmente, também foi apresentada a API para apoiar no desenvolvimento de aplicações 3D para engenharia de software por meio de visualização e interação com modelos UML 3D.

Vale ressaltar que o protótipo implementado também possui uma série de limitações. Do ponto de vista de detalhes e características dos modelos UML, o diagrama de sequência não fornece visualmente o tempo de vida de cada mensagem. Além disso, as classes que possuem relacionamento de herança não terão as setas características dos modelos 2D. Por questões técnicas na geração automatizada dos

[11] https://youtu.be/r2Glygcv0_M

modelos, esta característica visual não foi implementada. Também por questões técnicas, a disponibilização do código-fonte das classes do XMI foram geradas de forma semiautomática, ou seja, necessitou de uma interferência manual, diferente na geração dos digramas, os quais são automáticos.

CAPÍTULO 6 - AVALIAÇÃO DA ABORDAGEM

6.1. Introdução

Há evidências de que é possível perceber e compreender melhor os sistemas de software cada vez mais complexos, se eles são exibidos como objetos gráficos em um espaço tridimensional (WARE *et al.*, 1993).

A Engenharia de Software Experimental afirma que a validade de qualquer corpo de conhecimento deve ser avaliada para que esse conhecimento possa ser considerado científico (JURISTO & MORENO, 2013). Seguindo estes princípios, o protótipo da abordagem VisAr3D-Dynamic foi avaliado através de um estudo para verificar se a análise dinâmica por meio de Realidade Virtual de fato representa um diferencial na compreensão do comportamento dinâmico de sistemas de software orientados a objetos para os participantes durante a execução de tarefas predefinidas.

Este capítulo é organizado da seguinte forma: inicialmente foi apresentada a Introdução ao capítulo. Quanto ao objetivo e hipótese do trabalho, são definidos na Seção 6.2 e Seção 6.3, respectivamente. Na Seção 6.4, o escopo do estudo, bem como a seleção e arranjo dos participantes são apresentados. A Seção 6.5 descreve as tarefas executadas pelos participantes. A execução do estudo, tanto o piloto quanto o conduzido pelos participantes, são apresentados na Seção 6.6. Após a execução do estudo, os resultados e observações são discutidos na Seção 6.7. Na Seção 6.8, são realizadas discussões sobre a validade do estudo. Finalmente, a Seção 6.9 apresenta as considerações finais sobre o a avaliação da abordagem.

6.2. Objetivo

O propósito deste estudo é verificar a capacidade da abordagem VisAr3D-Dynamic de contribuir para a compreensão do comportamento dinâmico de software utilizando modelos UML e código-fonte em um ambiente virtual e tridimensional. Seguindo a abordagem GQM – Goal/Question/Metric (BASILI *et al.*, 1994), o objetivo do estudo é descrito conforme a Tabela 6.1.

Tabela 6.1 – Definição do objetivo do estudo de observação

Analisar	o uso de digramas UML e código-fonte em ambiente virtual e tridimensional disponibilizados pela abordagem VisAr3D-Dynamic
Com o propósito de	caracterizar
Em relação à	outra abordagem tradicional (*i.e.*, ferramenta Enterprise Architect)
Do ponto de vista de	mantenedores de software
No contexto da	compreensão do comportamento dinâmico de software orientado a objetos

6.3. Hipótese

É comum a análise dinâmica gerar um grande volume de dados, os quais são difíceis de serem compreendidos devido à complexidade e da quantidade das informações obtidas acerca da execução do software. Engenheiros, ao analisarem estes dados, precisam de ferramentas que os apoiem na execução de suas atividades de forma eficaz e eficiente, ou seja, que as informações sejam interpretadas corretamente e em tempo hábil.

A visualização é uma técnica bastante utilizada na compreensão de sistemas complexos. Por meio de representações gráficas, humanos podem obter melhores *insights* ao invés de analisar dados brutos em tabelas. No entanto, grande parte das ferramentas não exploram o meio pelo qual as visualizações são exibidas, ou seja, dispositivos de interface humano-computador, tais como monitores, sensores de gestos, ambientes de realidade virtual, dentre outros. Além disso, metáforas 3D são pouco utilizadas na análise dinâmica de software. Portanto, a ideia do estudo foi investigar o apoio oferecido pelo ambiente virtual, enquanto os mantenedores executam tarefas buscando compreender o software. Para tanto, foram utilizadas durante o estudo a comparação entre as ferramentas 2D (Enterprise Architect) e a 3D (protótipo VisaAr3D-Dynamic).

A hipótese deste estudo é que diagramas UML e código-fonte acessados por meio de um ambiente virtual e tridimensional podem auxiliar na compreensão do comportamento dinâmico de sistemas de software complexos em relação às ferramentas tradicionais com visualização bidimensional.

6.4. Definição do Estudo

O tipo de estudo adotado para a condução da avaliação é definido na literatura como estudo de observação ou observacional. Estudos observacionais são aqueles onde o participante realiza alguma tarefa enquanto é observado por um experimentador

(SHULL, CARVER & TRAVASSOS, 2001). Esse tipo de estudo tem a finalidade de coletar informações sobre como cada uma das tarefas foi realizada e, através dessas observações, obter uma compreensão de como o ambiente é efetivamente utilizado.

Por meio desse estudo, espera-se observar como mantenedores de software podem melhor compreender o comportamento dinâmico do software quando o protótipo da abordagem VisAr3D-Dynamic está disponível para apoiar o processo de compreensão. Mais especificamente, tentou-se observar, do ponto de vista qualitativo, se os participantes conseguiriam melhor compreender sistemas orientados a objetos sob determinado contexto de execução, interagindo com diagramas UML e código-fonte por meio de um ambiente virtual e tridimensional.

O estudo coloca cada participante no papel de mantenedor de software. O objetivo de um mantenedor é realizar alterações em artefatos, tal como código-fonte. Contudo, primeiramente, é necessário compreender as características e funcionalidades do software antes de realizar a alteração. No entanto, um cenário crítico em manutenção se dá quando o mantenedor não está familiarizado com o software, ou quando há poucos artefatos para guiar na compreensão. Neste sentido, o contexto do estudo é colocar o participante para compreender um cenário de execução por meio de diagramas de sequência, de classes, de pacotes UML e código-fonte.

O programa Odyssey foi escolhido para ser utilizado no estudo. Originalmente, Odyssey (WERNER *et al.*, 1999) é um projeto que visa explorar técnicas e ferramentas que apoiem a reutilização de software, permitindo a evolução de um ferramental de apoio. O software é um ambiente de reutilização baseado em modelos de domínio, provendo tecnologias de apoio à Engenharia de Domínio (ED), Linha de Produtos (LP) e Desenvolvimento Baseado em Componentes (DBC) (WERNER *et al.*, 1999). Ele serve como um arcabouço onde modelos conceituais e arquiteturas de software são especificados para domínios de aplicações específicos. O software contém 682 classes, 105 KLOC[12] e escrito na linguagem Java.

Optou-se como cenário de execução a ser analisado no experimento, o momento em que o usuário clica no ícone para criar um diagrama de *features*. Os dados deste cenário foram capturados por meio de instrumentação fornecida pela ferramenta

[12] Métrica de software que corresponde a mil linhas de código.

Enterprise Architect. Neste cenário, foram capturadas 37 chamadas de métodos, 14 *lifelines*, 11 classes e 4 pacotes.

6.4.1. Seleção e Arranjo dos Participantes

Os participantes do estudo são alunos e ex-alunos de pós-graduação da COPPE/UFRJ e alunos da IComp/UFAM. Todos possuem experiência prévia em modelagem UML e foram selecionados por conveniência. Foi proposto aos participantes um questionário com o objetivo de caracterizar sua formação do ponto de vista acadêmico, sua experiência profissional, entre outros, para analisar os dados e reduzir o viés, já que os participantes foram selecionados por conveniência. Considerou-se que estes indivíduos possuíam disponibilidade para participar do estudo. O questionário está disponível no Apêndice B (Caracterização do Participante).

Os participantes foram distribuídos, aleatoriamente, em dois grupos de trabalho: Grupo A e Grupo B. Os participantes do Grupo A utilizavam primeiramente a Configuração A e, em seguida, a Configuração B. Já os participantes do Grupo B utilizavam a Configuração B e depois a Configuração A. Cada grupo fez apenas uma passagem do experimento de modo a evitar os efeitos da aprendizagem.

Configuração A: Utilização da ferramenta EA como visualizador de diagramas UML e código-fonte para resolução de cinco tarefas propostas. A Figura 6.1 mostra uma tela da EA utilizada durante o experimento.

Configuração B: Utilização do protótipo VisAr3D-Dynamic como visualizador de diagramas UML e código-fonte para resolução de cinco tarefas propostas (semelhantes às tarefas da configuração A). A Figura 6.2 mostra uma tela do VisAr3D-Dynamic utilizada durante o experimento.

Por conveniência, os participantes realizaram o estudo remotamente. Durante todo o experimento, foi mantido o contato por meio de áudio através do software Skype (SKYPE, 2017). O protótipo foi instalado no computador de cada participante e o EA foi acessado remotamente através do software TeamViewer (TEAMVIEWER INC., 2017). Durante a execução das tarefas, a interação com o protótipo foi observada através do Skype (função de compartilhar tela) e no EA, como os participantes conectaram a um computador configurado para o experimento e acessível ao pesquisador, não precisou de software para acompanhar a realização das tarefas. Os

formulários dos Anexos A, B, C e D foram disponibilizados em formato digital e o tutorial do protótipo e do EA no formato de apresentação.

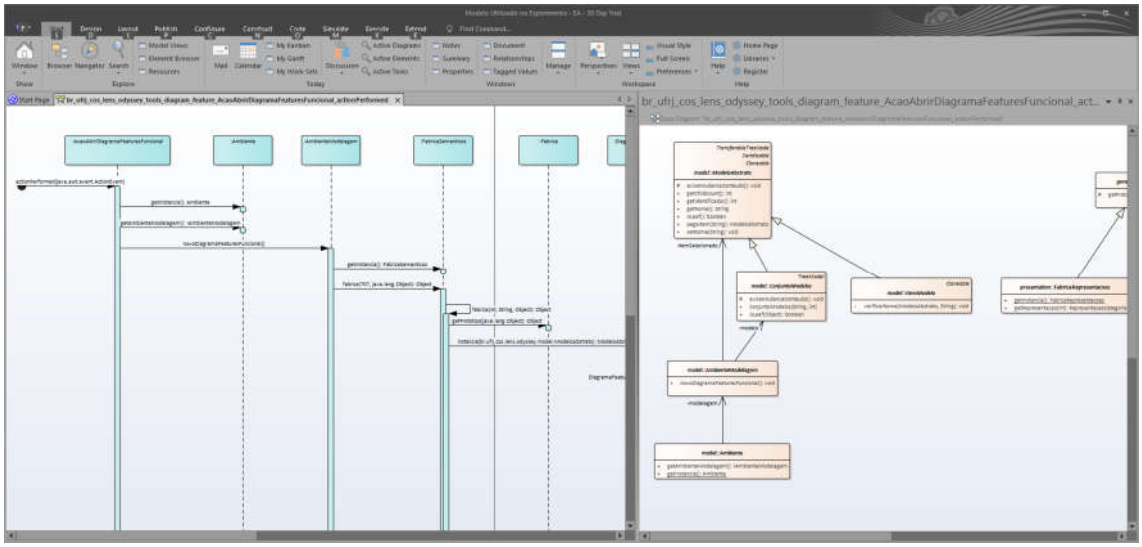


Figura 6.1 – Tela da EA

Os participantes ficaram à vontade para fazerem perguntas de esclarecimento, enquanto acompanhados pelo pesquisador responsável, que fazia anotações. O arranjo dos participantes aos grupos foi dado pela alternância entre o Grupo A e B de acordo com a ordem de participação no estudo. Por exemplo, o primeiro participante foi alocado ao Grupo A, o segundo ao Grupo B, o terceiro ao Grupo A e assim sucessivamente. Portanto, a alocação dos participantes aos grupos foi aleatória.

6.5. Tarefas

Ambos os grupos A e B foram convidados a resolver 10 tarefas como mantenedores de software (Apêndice C). As tarefas representaram as principais atividades de compreensão de sistemas orientados a objetos, utilizando para isto a análise de um diagrama de sequência, um diagrama de classes, um de pacotes e código-fonte de cada classe, que representa um cenário de execução do software Odyssey.

As tarefas do experimento foram elaboradas com o intuito de corresponderem às principais atividades durante a compreensão de programas. Para tanto, foram utilizadas algumas atividades gerais para compreensão de programas propostas por PACIONE *et al.*, (2003) (Tabela 6.2), com o intuito de guiar na concepção das tarefas. A Tabela 6.3 lista o tipo de tarefa e suas as atividades gerais relacionadas. Para o protótipo e o EA foram elaboradas cinco tarefas baseadas nos tipos da Tabela 6.3, porém com valores diferentes para diminuir o viés de aprendizagem do cenário de execução do Odyssey.

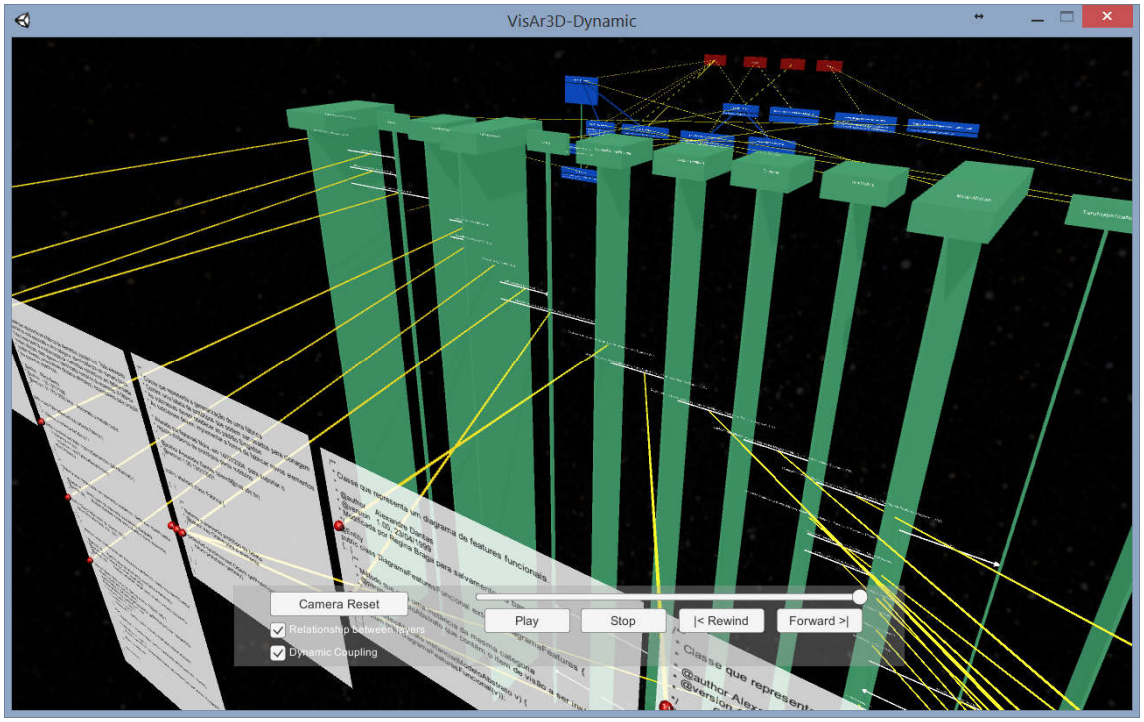


Figura 6.2 – Tela do VisAr3D-Dynamic

Tabela 6.2 – Principais atividades de compreensão de programas (PACIONE *et al.*, 2003)

Atividade	Descrição
A1	Investigar a funcionalidade (ou parte) de um sistema
A2	Modificar a funcionalidade de um sistema
A3	Investigar a estrutura interna de um artefato
A4	Investigar dependências entre os artefatos
A5	Investigar interações dinâmicas do sistema
A6	Investigar como um artefato é usado
A7	Investigar padrões na execução do sistema
A8	Acessar a qualidade do *design* do sistema
A9	Entender o domínio do sistema

Tabela 6.3 – Tipos de tarefas realizadas pelos indivíduos

Tarefa	Atividade	Descrição
T1	A7, A8	Identificar o pacote que possui maior ou menor contribuição na execução de acordo com a quantidade de classes executadas.
T2	A1, A3, A4, A5, A6	Identificar o método construtor da classe e compreender por que o próximo método é executado.
T3	A1, A3, A4, A5, A6, A9	Localizar e analisar o retorno de um método de acordo com o parâmetro informado.
T4	A5, A7, A8	Identificar o grau de acoplamento dinâmico dos objetos no diagrama de sequência.
T5	A1, A3, A4, A5, A6, A9	Identificar métodos que ocorrem polimorfismo.

6.6. Execução do Estudo

6.6.1. Estudo Piloto

Antes da realização do estudo com os participantes selecionados, foi efetuado um piloto do estudo de observação em janeiro de 2017. O participante convidado, professor do IF Sudeste MG e mestrando pela UFV, realizou todos os procedimentos mencionados, a fim de que fosse possível verificar a adequação do plano de estudo e seus instrumentos para efetivação com os demais participantes. A partir da execução deste piloto, foi possível notar que uma tarefa estava ambígua e o tempo da execução total das tarefas estava muito alto. Como solução, esta tarefa foi retirada do estudo e as restantes foram adaptadas. Além disso, notou-se que o tutorial de utilização do protótipo poderia ser melhorado, visto que é comum ferramentas UML em 2D do que em 3D. Também ajudou a estimar o tempo necessário em cada tarefa, bem como deixar o ambiente configurado.

6.6.2. Estudo de Observação

A condução do estudo propriamente dito, foi realizada remotamente, portanto, dois computadores foram utilizados para o experimento: um do participante com o protótipo instalado e outro com o EA instalado no computador do pesquisador. Cada sessão do estudo utilizou um único participante, durando cerca de 2 horas.

Inicialmente, cada participante foi informado sobre o experimento através do Formulário de Consentimento (Apêndice A), tomando conhecimento sobre o seu objetivo, formato e termo de confidencialidade. A partir da concordância em participar do experimento, o participante preencheu o Formulário de Caracterização (Apêndice B). Este questionário avaliou o seu nível de conhecimento e experiência. Estas informações garantiram que os mesmos estavam aptos a executar o estudo e também serviram para interpretar os resultados obtidos por cada um dos participantes. O preenchimento deste formulário durou em média 8 minutos e 4 segundos.

Posteriormente, foi disponibilizado um tutorial de treinamento com imagens e textos no formato de apresentação com uma breve explicação das principais funcionalidades de cada ferramenta e os elementos gráficos disponíveis. O tempo de leitura do tutorial do protótipo VisAr3D-Dynamic durou em média 10 minutos e 19 segundos, contendo 85 *slides*, e o tutorial do EA durou em média 3 minutos e 30 segundos, contendo 30 *slides*.

Antes de começar a resolver as tarefas com uma ferramenta, cada participante teve alguns instantes para explorá-la com o objetivo de familiarizar-se um pouco mais. Em média, os participantes utilizaram o EA por 4 minutos e 41 segundos e 5 minutos e 51 segundos no protótipo. Um guia do protótipo também foi disponibilizado para sanar algumas dúvidas durante o experimento.

Ao final do treinamento, foi disponibilizado um formulário com todas as tarefas que deveriam ser resolvidas, 5 tarefas com a ferramenta EA e 5 tarefas com o protótipo VisAr3D-Dynamic. A resolução das tarefas durou em média 4 minutos e 4 segundos. Um modelo do formulário de tarefas está disponível no Apêndice C. Para cada tarefa, foi solicitado, ainda, que o participante a avaliasse segundo um nível de dificuldade: fácil, médio e difícil. Em seguida, o pesquisador anotava o tempo gasto para sua resolução.

Ao final da resolução das tarefas, os participantes responderam a um Questionário de Avaliação (Apêndice D), abrangendo questões de percepção do participante: quanto à qualidade das tarefas, qualidade dos resultados, questões sobre a própria experiência (tempo suficiente, adequação das tarefas) e comparação entre as duas ferramentas. O preenchimento deste questionário durou em média 18 minutos e 7 segundos. Finalmente, o material resultante de cada participante foi recolhido.

6.7. Resultados e Observações

6.7.1. Caracterização dos Participantes

Conforme dito anteriormente, os participantes do estudo são alunos e ex-alunos de pós-graduação da COPPE/UFRJ e alunos de pós-graduação do IComp/UFAM, os quais foram selecionados por conveniência. Não houve nenhum tipo de compensação para os mesmos. No total, 10 indivíduos participaram do estudo, entre eles, 1 doutor, 7 doutorandos e 2 mestrandos. A Tabela 6.4 apresenta a distribuição dos grupos em relação ao número de participantes.

Tabela 6.4 – Número de participantes por grupo

	Grupo A			**Grupo B**		
Grupo de pessoas	**Doutores**	**Doutorandos**	**Mestrandos**	**Doutores**	**Doutorandos**	**Mestrandos**
Qtde.	1	4	0	0	3	2

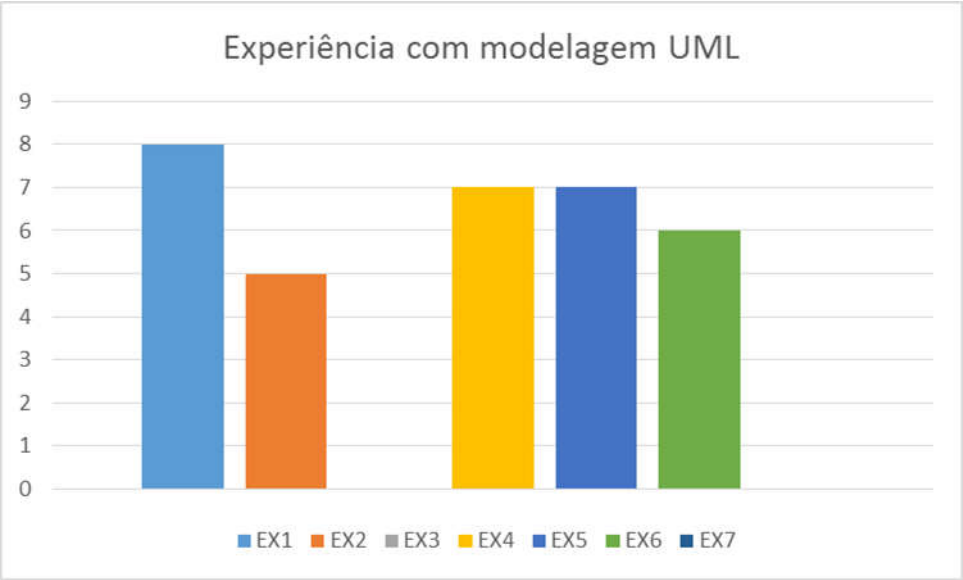


Figura 6.3 – Número de participantes para cada nível de experiência com modelagem UML

Tabela 6.5 – Opções para o grau de experiência com modelagem UML

Opção	Descrição
EX1	Já li material sobre modelagem UML
EX2	Já participei de um curso sobre modelagem UML
EX3	Nunca fiz uma modelagem UML
EX4	Já fiz modelagem UML para uso próprio
EX5	Já fiz modelagem UML como parte de uma equipe, relacionada a um curso
EX6	Já fiz modelagem UML como parte de uma equipe na indústria
EX7	Outro

A Figura 6.3 apresenta o grau de experiência dos participantes com modelagem UML. Embora no estudo não contenha tarefas para a criação de modelos UML, adotou-se o critério da experiência em modelagem para continuar a participar do estudo. Nota-se que todos possuem alguma experiência com modelos UML. A diferença entre os participantes que já fizeram modelagem para uso próprio, relacionado a um curso e os que fizeram modelagem como parte de uma equipe na indústria é muito tênue. A Tabela 6.5 lista as opções de grau de experiência apresentadas aos participantes.

Como o foco do estudo é a compreensão do comportamento dinâmico de software, buscou-se investigar a experiência de cada participante na leitura de diagramas de sequência para entender a troca de mensagens entre objetos. Conforme demostra a Tabela 6.4, 90% dos participantes já leram sobre o assunto. Além disso, 60% dos participantes já necessitaram entender o comportamento no diagrama para uso próprio, bem como parte de uma equipe relacionada a um curso. A Tabela 6.6 lista as

opções de experiência na compreensão de diagramas de sequência UML apresentadas aos participantes.

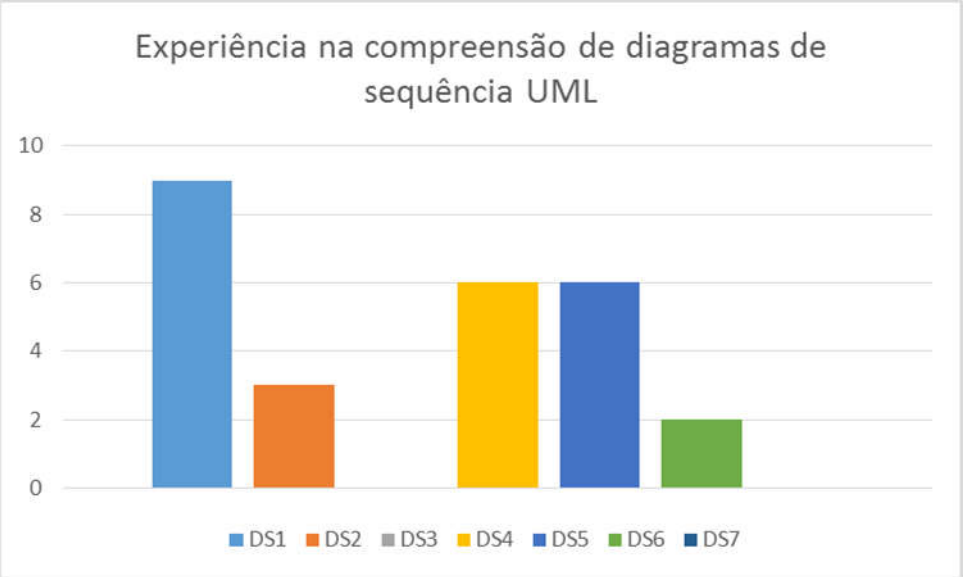


Figura 6.4 – Número de participantes para cada nível de experiência na compreensão da troca de mensagens entre objetos no diagrama de sequência UML

Tabela 6.6 – Opções para o grau de experiência na compreensão de diagramas de sequência UML

Opção	Descrição
DS1	Já li material sobre este assunto
DS2	Já participei de um curso sobre este assunto
DS3	Nunca precisei entender sobre este assunto
DS4	Já precisei compreender a troca de mensagens entre objetos no diagrama de sequência para uso próprio
DS5	Já precisei compreender a troca de mensagens entre objetos no diagrama de sequência como parte de uma equipe relacionada a um curso
DS6	Já precisei compreender a troca de mensagens entre objetos no diagrama de sequência como parte de uma equipe na indústria
DS7	Outro

Do ponto de vista da experiência com algumas tecnologias, tais como orientação a objetos, padrões de projeto, Java e outras linguagens, a maioria dos participantes já utilizaram estas tecnologias na indústria. Conforme demonstra a Figura 6.5, 60% dos participantes já usaram conceitos de orientação a objetos. Os 45% dos participantes conhecem a linguagem Java e 40% tiveram contato com padrões de projeto e com outras linguagens também na indústria. Decidiu-se obter este tipo de informação para apoiar na análise qualitativa do estudo, pois algumas tarefas necessitam de conhecimentos, principalmente, sobre orientação a objetos e Java. A Tabela 6.7 lista a escala do grau de experiência apresentada aos participantes.

Mais da metade dos participantes já tiveram contato com modelagem UML (Figura 6.3) e com orientação a objetos (Figura 6.5). Tal caracterização é importante por verificar que os participantes têm por hábito lidar com problemas reais.

6.7.2. Análise do Desempenho das Tarefas

Como mencionado anteriormente, foram apresentadas 5 tarefas a serem realizadas em cada ferramenta pelos 10 participantes do estudo. Além disso, 5 participantes foram agrupados no Grupo A e o restante no Grupo B.

A Figura 6.6 demostra a soma de acertos de cada grupo por tarefa em relação à ferramenta. Nela, é apresentada a quantidade de acertos de cada tarefa por grupo. Pode-se notar que o Grupo B possui maior discrepância de acertos em relação ao Grupo A, nas tarefas T3 e T4 da EA. A diferença entre os grupos são de 2 acertos. Pede-se na T3, para localizar e analisar o retorno de um método de acordo com o parâmetro informado. Os participantes do Grupo B não acertaram esta tarefa, ao passo que 2 participantes do Grupo B conseguiram acertar. Em relação à T4, que diz respeito à métrica de acoplamento dinâmico, todos os participantes do Grupo B e 3 participantes do Grupo A acertaram a tarefa. Analisando esta tarefa executada no protótipo, T4 obteve diferença de 2 acertos em relação aos grupos.

Ao comparar cada tarefa e a sua equivalente, pode-se notar que a quantidade de acertos no protótipo aumentou consideravelmente. Os participantes dos dois grupos obtiveram o mesmo resultado na T1 nas duas ferramentas. Enquanto que na EA foram obtidos 3 acertos e no protótipo, todos os participantes do estudo acertaram a tarefa. No caso da T2, os participantes do Grupo A obtiveram a mesma quantidade de acertos em ambas as ferramentas. Já o Grupo B, aumentou 1 acerto utilizando o protótipo.

A tarefa T3 do protótipo obteve o melhor resultado do Grupo B em relação à EA. Nenhum participante deste grupo ao utilizar a EA conseguiu efetivamente localizar e analisar o retorno de um método de acordo com o parâmetro informado. No protótipo, 3 participantes conseguiram responder adequadamente o que se pedia. Já no Grupo A, 3 participantes acertaram a tarefa utilizando o protótipo e 2 participantes acertaram na EA.

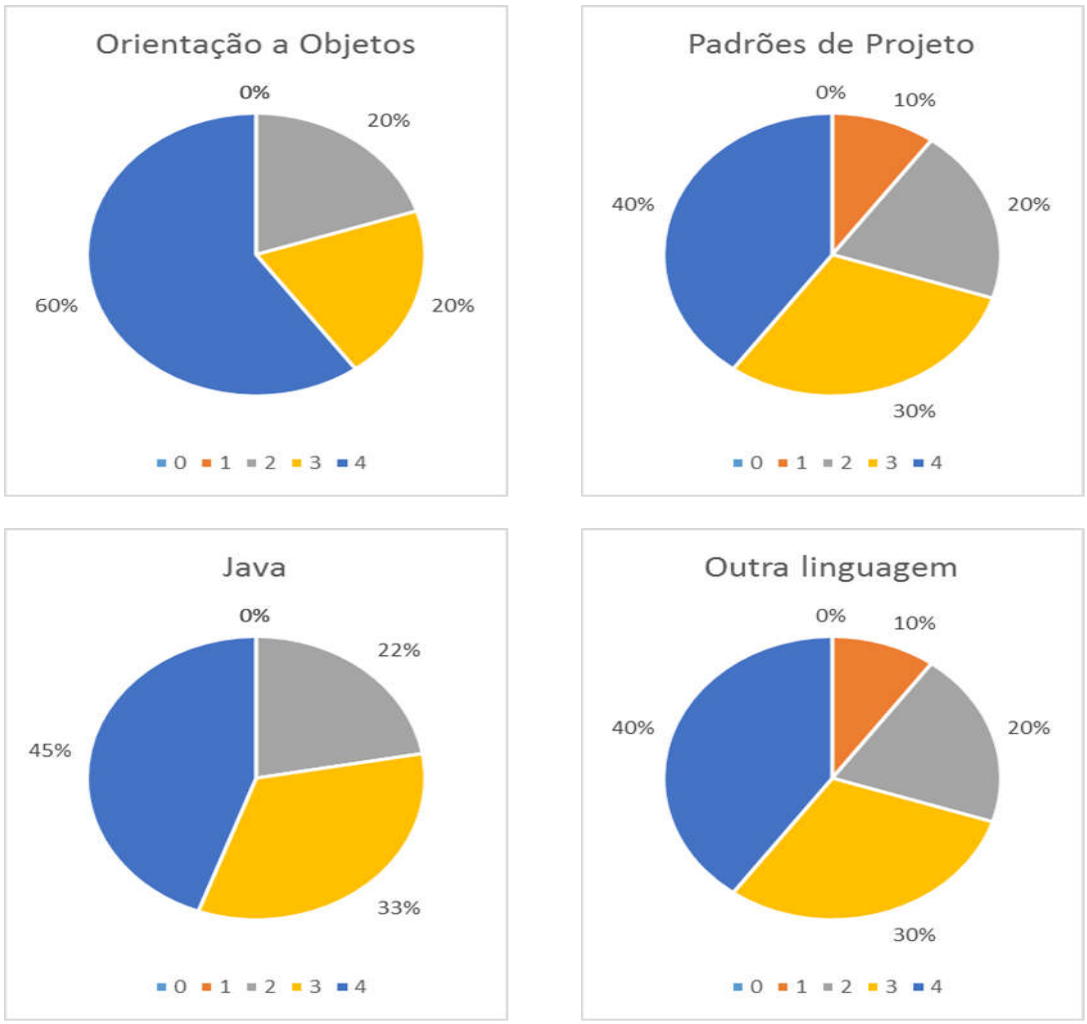


Figura 6.5 – Graus de experiência em orientação a objetos, padrões de projeto, Java e em outras linguagens

Tabela 6.7 – Opções para o grau de experiência em orientação a objetos, padrões de projeto, Java e em outras linguagens

Opção	Descrição
0	Nenhum
1	Estudei em aula ou em livro
2	Pratiquei em projetos em sala de aula
3	Usei em projetos pessoais
4	Usei em projetos na indústria

O único caso crítico, do ponto de vista do objetivo da proposta da abordagem, foi identificado na T4 do protótipo pelos participantes do Grupo B. Todos os participantes acertaram a tarefa utilizando a EA. Contudo, apenas 1 participante deste grupo conseguiu efetivamente realizar a tarefa no protótipo. Contudo, os participantes do Grupo A mantiveram a quantidade de acertos nas duas ferramentas. Pela ferramenta

EA não fornecer a informação de métricas de acoplamento dinâmico, foi disponibilizado um material adicional que listava cada objeto com sua métrica. De acordo com um participante, ao final do estudo, comentou que a forma como foi apresentada as métricas na EA foi desleal e, ainda, sugeriu que, para ficar mais justo, poderia ter fornecido a fórmula para calcular a métrica de cada objeto manualmente.

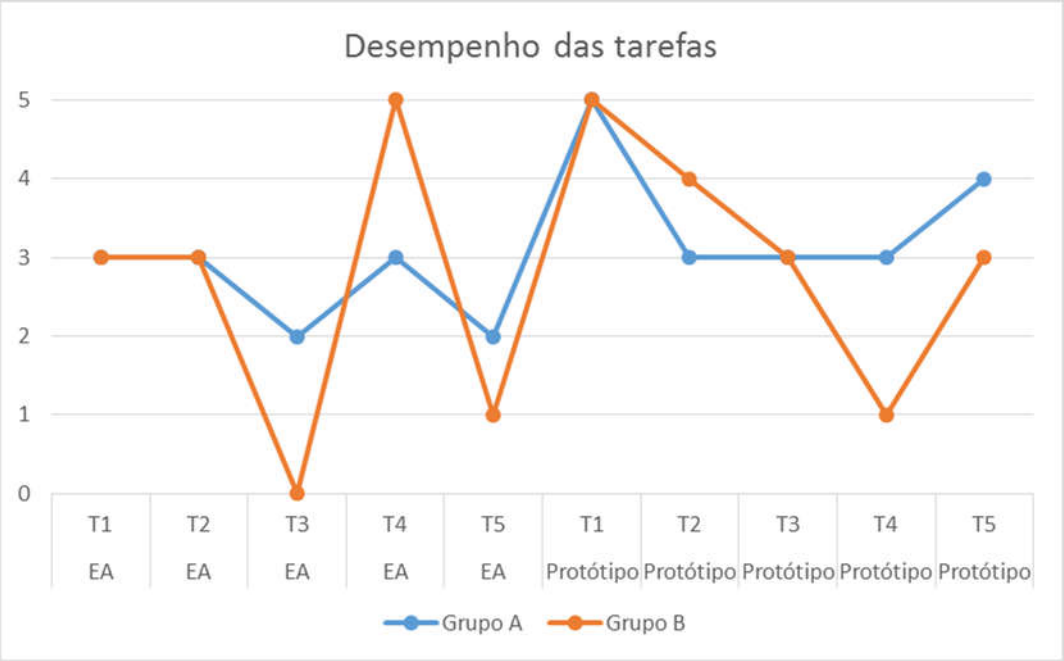


Figura 6.6 – Acertos dos participantes em relação à ferramenta e ao grupo alocado

Durante a execução da T4 no protótipo, foi observado que alguns participantes tinham dificuldade de se orientar e visualizar a profundidade de cada objeto. Além disso, alguns participantes perguntaram se a largura tinha influência no valor da métrica. Neste caso, o pesquisador não se pronunciou, visto que esta informação estava no tutorial do protótipo. Pôde-se observar também que para alguns participantes proporcionou ambiguidade ao se perguntar "o(s) objeto(s) com alto acoplamento dinâmico". Por ter alguns objetos com o mesmo acoplamento dinâmico baixo, alguns participantes imaginaram que a resposta desta tarefa era os objetos restantes. No entanto, havia somente um objeto com o maior acoplamento dinâmico no diagrama de sequência, o qual era esperado como resposta. Optou-se por colocar o "s" em parênteses para reduzir o viés de influenciar na resposta.

Por fim, a T5 obteve aumento de dois acertos pelos dois grupos no protótipo. O objetivo desta tarefa foi identificar métodos que ocorrem polimorfismo. No Grupo A, 2 participantes acertaram na EA, ao passo que no protótipo foram 4 participantes. No caso

do Grupo B, 1 participante obteve êxito na EA e 3 participantes conseguiram realizar esta tarefa no protótipo.

A Figura 6.7 mostra a quantidade de acertos dos participantes por ferramenta. O participante B não consta no gráfico, pois desistiu do estudo. Como pode ser observado, 7 participantes obtiveram mais acertos no protótipo em relação à EA. Em meio aos 10 participantes, destaca-se o participante F, por acertar somente 2 de 10 tarefas e destas 2, ser no protótipo. Quando perguntado em qual ferramenta foi utilizada para apoiar a compreensão de troca de mensagens entre objetos, o participante respondeu que nunca precisou de realizar este tipo de procedimento. Além disso, a modelagem UML só foi utilizada no contexto de curso e leitura sobre UML. Apesar do estudo não focar em ensino-aprendizagem de modelos UML, o protótipo obteve vantagem em relação ao participante que possui falta de prática e de contato com cenários reais.

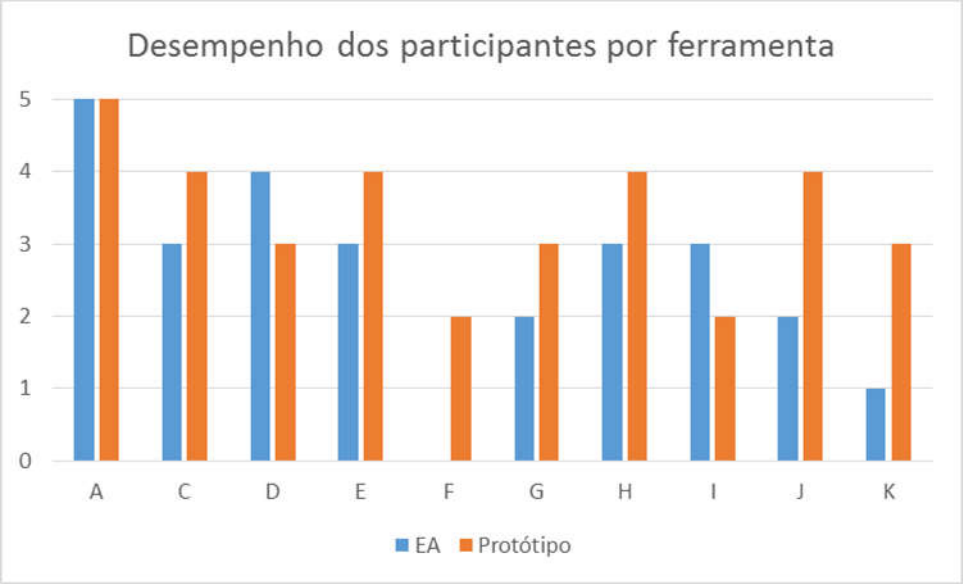


Figura 6.7 – Acertos dos participantes por ferramenta

Por fim, a Figura 6.8 demonstra a soma de acertos de cada tarefa em relação às ferramentas. Nota-se que o protótipo obteve pontuação maior do que a EA em 3 dos 5 tipos de tarefas. Uma tarefa ficou empatada e a outra a EA obteve maior vantagem. Como já mencionado anteriormente no caso da T4, a forma como foi disponibilizada a informação da métrica de acoplamento dinâmico na EA, pôde ter influenciado na resolução da tarefa.

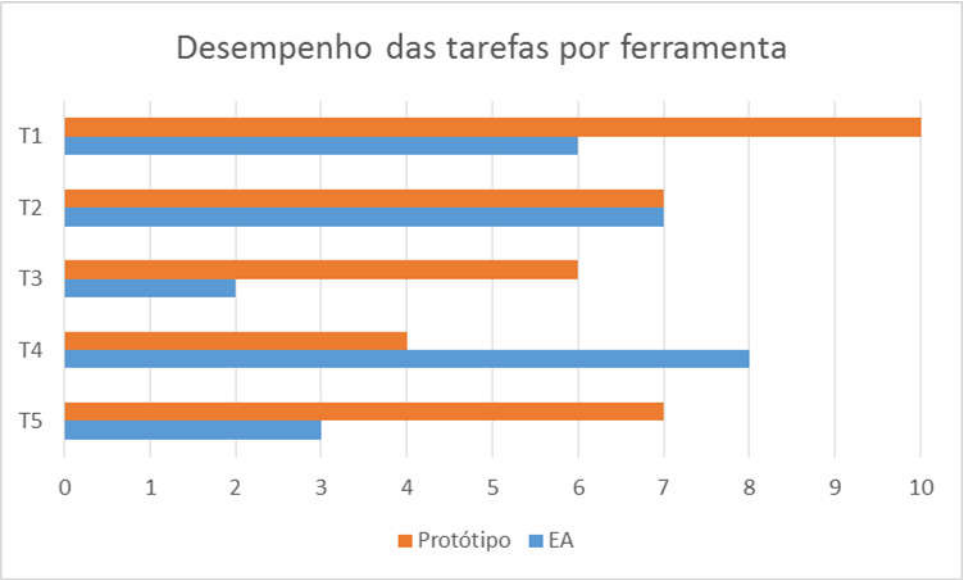


Figura 6.8 – Acertos das tarefas por ferramenta

No que tange à identificação de um determinado método e investigar o motivo de o próximo método ser executado, tanto no protótipo quanto na EA, obtiveram a mesma pontuação na T2. No entanto, os participantes conseguiram desempenhos satisfatórios em T1, T3 e T5, quando utilizaram o protótipo. Quanto à identificação de pacotes proposto pela T1, ao longo do estudo observou-se uma certa facilidade de percepção no protótipo devido ao número de *links* que cada pacote continha. Apesar de demorarem um pouco mais do que na EA, conseguiam responder com exatidão. Por ser o primeiro contato com o ambiente, esperava-se um certo atraso.

A T3 exigia um pouco mais na exploração de informação entre digramas e código-fonte, principalmente no comentário existente na implementação do método. Na Figura 6.9 e Figura 6.10, respectivamente, são demonstrados os níveis de dificuldade que cada participante julgou para cada tarefa. Pode-se notar que 80% dos participantes marcaram T3 como fácil de ser realizada. Contudo, 50% julgaram esta tarefa difícil na EA. Observou-se que o *link* entre o diagrama e código-fonte apoiou a resolução desta tarefa, visto que necessitava alternar entre os diferentes artefatos.

Finalmente, na T5 os 30% dos participantes julgaram como fácil e os outros 30% como difícil. Já os 40% restantes assinalaram como uma tarefa com dificuldade mediana. No entanto, 70% deles conseguiram efetuar corretamente a tarefa no protótipo. Alguns participantes durante o estudo expressaram verbalmente a dificuldade de

navegabilidade e também o fato de o diagrama de classe não explicitar herança através de setas iguais ao tradicional, mas por cores diferenciadas dos relacionamentos comuns.

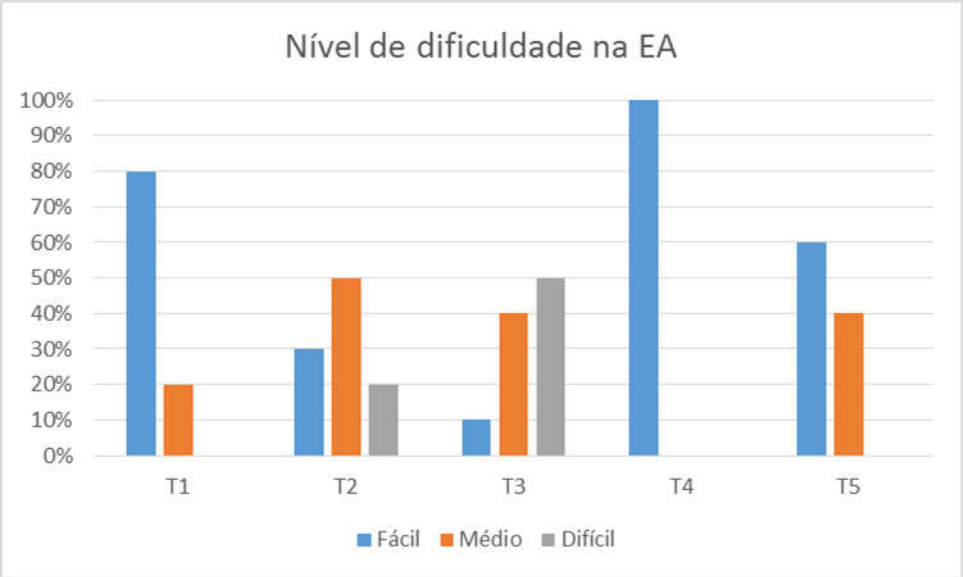


Figura 6.9 – Nível de dificuldade de cada tarefa na ferramenta EA

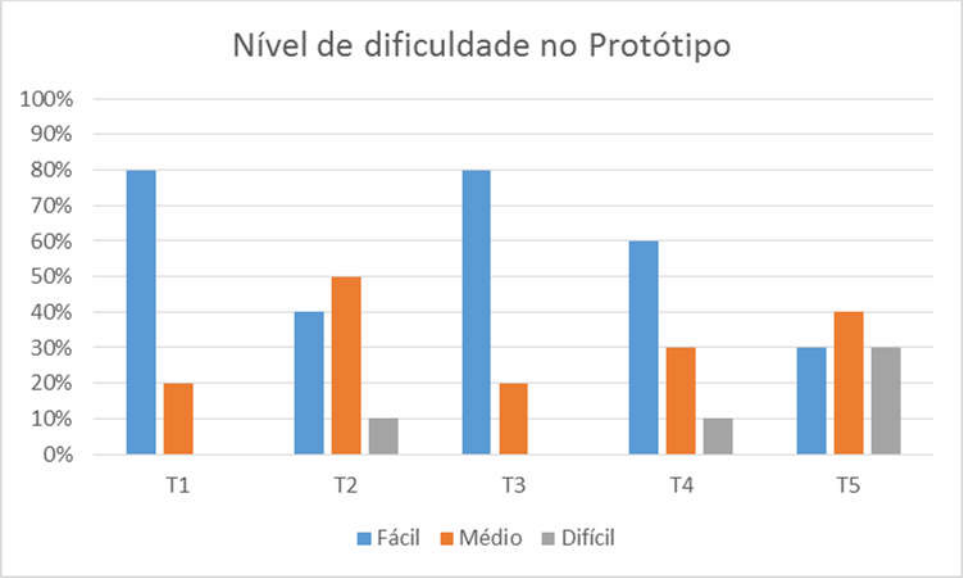


Figura 6.10 – Nível de dificuldade de cada tarefa no Protótipo

6.7.3. Análise do Questionário de Avaliação

Depois de executar as tarefas do estudo, os participantes preencheram um questionário de avaliação com perguntas referentes a execução do estudo e questões genéricas em relação às duas abordagens adotadas. Com relação a abordagem VisAr3D-Dynamic os participantes se mostraram bastante satisfeitos com os resultados obtidos e com a facilidade em responder perguntas que, tomando como base suas experiências de

modelagem UML e maior familiaridade com ferramentas 2D, não seriam facilmente respondidas. A Tabela 6.8 lista algumas das perguntas feitas no questionário de avaliação aos participantes (Apêndice D) ordenadas de forma decrescente em relação à vantagem da utilização do 3D. Alguns comentários desses participantes serão apresentados e discutidos a seguir.

Tabela 6.8 – Contribuição por ferramenta segundo alguns tópicos

Id	Questão do questionário de avaliação	2D	3D	Não se aplica
14	Suporte à compreensão do comportamento dinâmico	0%	100%	0%
16	Melhor engajamento na exploração de informações acerca do comportamento dinâmico	0%	100%	0%
6	Apoio à exploração dos modelos	10%	90%	0%
7	Apoio à resolução das tarefas	10%	90%	0%
10	Facilidade na exploração de informações entre digramas e código	0%	90%	10%
11	Facilidade na exploração de informações a cada mensagem executada	0%	90%	10%
15	Melhor percepção de informações relacionadas entre os modelos e o código	10%	90%	0%
9	Disponibilização de ambiente mais intuitivo	10%	80%	10%
12	Facilidade de obtenção de informações sobre aspectos de qualidade de software	0%	80%	20%
5	Apoio à compreensão da rota de execução através do diagrama de sequência UML	10%	60%	30%
8	Redução da complexidade da visualização	40%	50%	10%
13	Facilidade no entendimento sobre herança e polimorfismo quando o software é executado	20%	50%	30%

(*) “Não se aplica” foi considerado pelos participantes com dúvida entre uma ferramenta e outra, e como nenhuma das duas opções.

Em relação ao suporte à compreensão do comportamento dinâmico, todos os 10 participantes concordaram que na abordagem 3D é possível uma melhor compreensão do comportamento dinâmico em relação ao 2D. Um deles comentou “é possível visualizar claramente essa troca, característica do comportamento dinâmico do diagrama de sequência”; outros explicitaram que o diferencial foi a execução passo a passo. Também com relação ao engajamento na exploração de informações acerca do comportamento dinâmico, 100% dos participantes concordaram que o 3D apoia melhor esta característica. “Ajudou com a disponibilidade de informações e troca entre camadas seguindo as relações”, comentou um dos participantes. Outro participante disse que se sentiu motivado e engajado na ferramenta 3D, pelo fato do controle da execução que é fornecido.

Os 90% dos participantes do estudo julgaram que o 3D melhor apoiou na exploração dos modelos. Dentre os comentários feitos, um participante mencionou que, apesar de algumas dificuldades, possivelmente a abordagem pode ser aplicada também com outro foco. Segue o comentário na íntegra: “achei divertido, mas tive alguns problemas com a navegabilidade que para mim não era tão intuitivo, talvez por não estar acostumado com os comandos. Mas vejo que o *link* com os diferentes diagramas e código-fonte pode ser uma boa ferramenta para ensino de modelagem para alunos de graduação que frequentemente têm dificuldade de visualizar o mapeamento modelo-código”. A abordagem proposta nesta dissertação de mestrado estende a abordagem VisAr3D, a qual objetiva apoiar o ensino e aprendizagem de sistemas complexos. Durante as etapas iniciais desta pesquisa, a ideia era apoiar também o ensino, porém do ponto de vista do comportamento dinâmico de sistemas. No entanto, foi observado que havia problemas de sobrecarga cognitiva durante a compreensão do comportamento dinâmico de sistemas complexos, visando evoluir e modificar estes sistemas. Portanto, o foco da abordagem foi dado no contexto de manutenção, mas como trabalhos futuros, poderá ser realizado um novo estudo com o objetivo de buscar evidências acerca da abordagem em relação ao ensino e aprendizagem de modelagens de sistemas.

Ainda nesta questão, outro participante disse que havia receios do 3D em relação à exploração dos modelos. Segundo o participante, “de fato o ambiente em 3D favorece à análise. Eu tinha receios quanto a isso, mas fiquei surpreso que realmente girar, aplicar *zoom*, navegar em camadas ajudou muito”.

Outra vantagem do 3D (90%) em relação ao 2D (10%), foi relacionada à facilidade na exploração de informações a cada mensagem executada. Dois participantes comentam a importância e facilidade de entender o comportamento com a execução passo a passo dos modelos. Um participante comentou: “acabei não utilizando a *timeline* do modelo 3D. Eu deixei o modelo todo ‘executado’ no último passo e respondi as perguntas olhando apenas para o modelo completo. Mas agora, lendo essa pergunta, entendo que seria mais fácil responder algumas perguntas 'executando' passo a passo o modelo”. Já o outro participante disse que “fluxos de execução é muito melhor de ser percebido pelo 3D, talvez pelas funcionalidades do protótipo, não muito pelas características tridimensionais”. Este participante deixa claro que não é o fato de ser 3D, mas pela funcionalidade que a abordagem 3D proporciona, a qual é ausente no 2D.

Em relação à redução da complexidade da visualização, os participantes ficaram divididos entre a ferramenta 2D e 3D. Os 50% dos indivíduos que realizaram o estudo optaram pelo 3D; 40% deles escolheram 2D e 10% como "não se aplica". Muito interessante são os comentários feitos: "o 3D apresenta muita informação depois de completo. Acho que para uma visualização do todo (de tudo executado ao mesmo tempo) ele pode ficar confuso. Contudo, para o passo a passo (que eu acho que está rápido) creio que o 3D é melhor. Existem benefícios com os dois tipos de abordagens, visualizar o fluxo e o *link* com os diferentes diagramas fica muito melhor no 3D". Outro participante disse que "a complexidade dos diagramas é independente de ferramenta. Podem ser muito grandes e complexos, isso depende do sistema e do conhecimento do usuário". Por outro lado, "apesar de necessário um maior tempo de teste, a visualização 3D se torna muito mais fácil de navegar após um tempo, enquanto que na 2D é necessária a troca de abas. No 3D é possível visualizar os pacotes, classes e diagramas mais facilmente", segundo outro participante.

Aos participantes também foi perguntado sobre o grau de relevância de cada funcionalidade do protótipo, segundo seus pontos de vista. A Tabela 6.9 lista as funcionalidades ordenadas de forma decrescente em relação à escala Likert, que varia de menor relevância, representado com valor 1, e de maior relevância, com o valor 5.

Tabela 6.9 – Grau de relevância para cada função do protótipo

Função do protótipo	**Escala**				
	1	**2**	**3**	**4**	**5**
Go to the Lifeline	0%	0%	10%	0%	90%
Relationship between layers	0%	0%	10%	10%	80%
Go to the Class	0%	0%	10%	10%	80%
Camera Reset	0%	0%	20%	0%	80%
Go to the Code	0%	0%	20%	0%	80%
Go to the Message in Code	0%	0%	20%	0%	80%
Go to the Package	0%	0%	20%	10%	70%
Dynamic Coupling	10%	0%	10%	10%	70%
Go to the Message in Lifeline	0%	10%	20%	0%	70%
As teclas A, S, D e W	20%	0%	10%	0%	70%
O botão esquerdo do mouse para movimentação da câmera	0%	0%	40%	0%	60%
O botão do meio do mouse para zoom	0%	10%	30%	0%	60%
Stop	10%	0%	20%	20%	50%
As teclas ←↓→↑	10%	20%	10%	10%	50%
Go to the Next Breakpoint	10%	10%	20%	20%	40%
Play	0%	0%	20%	50%	30%
Slider (botão deslizante para esquerda e direita)	0%	10%	10%	50%	30%
Rewind	0%	10%	30%	30%	30%
Forward	0%	10%	30%	30%	30%

O recurso mais utilizado pelos participantes foi "*Go to the Lifeline*", disponível por meio do Menu de Contexto. Já era de se esperar que esta funcionalidade seria uma das mais relevantes. Como o objetivo do estudo, bem como da abordagem, é apoiar a análise dinâmica, consequentemente a visualização de *lifelines* seria frequente, visto que o único diagrama comportamental é o de sequência. A partir da segunda funcionalidade mais relevante, as cinco funcionalidades seguidas tiveram relevância máxima entre os 80% dos participantes. Dentre elas, 3 são do Menu de Contexto (*Go to the*...), uma é sobre a visualização do *link* entre as perspectivas (*Relationship between layers*) e a outra funcionalidade está relacionada no auxílio ao usuário caso ocorra desorientação no ambiente virtual.

6.8. Validade

É comum que haja questões que possam impactar ou limitar a validade dos resultados dos estudos. Estas questões são denominadas ameaças à validade (WOHLIN *et al.*, 1999; TRAVASSOS *et al.*, 2002). Este estudo preliminar foi executado a fim de observar o uso controlado da abordagem desenvolvida e compará-la com uma abordagem tradicional encontrada no contexto de análise dinâmica. Os resultados obtidos a partir das observações e da avaliação dos participantes nos ajudaram a entender as limitações da abordagem e os pontos que precisam ser melhorados. No entanto, devido às restrições deste estudo, os resultados obtidos não podem ser generalizados.

A escolha dos participantes foi realizada de forma não aleatória, por conveniência, com pessoas conhecidas do experimentador. Os participantes têm perfil variado com relação à experiência em modelagem UML de sistemas orientados a objetos. Aliado a isso, o número reduzido de participantes também se coloca como outra restrição do estudo de observação conduzido.

Como o estudo foi realizado a distância, alguns participantes tiveram por questões de segundos a falta de conectividade com a Internet. Além disso, o pequeno atraso no tempo de resposta da interação com a ferramenta EA devido ao acesso remoto foi inevitável. No entanto, nenhum participante comentou sobre este aspecto. Uma das motivações da abordagem é o apoio à compreensão de um grande volume de dados gerados a partir da execução de sistema de larga escala devido ao problema de escalabilidade das visualizações. Contudo, os modelos utilizados, do ponto de vista de

quantidade de elementos de modelagem, podem não ser considerados complexos. Por outro lado, o fato de ser um sistema que o participante não esteja familiarizado, de certa forma incide complexidade. Além disso, talvez modelos maiores poderiam interferir negativamente no estudo, demorando na resolução de cada tarefa, por exemplo.

Para realizar algumas justificativas na análise dos dados, as informações de caracterização que cada participante forneceu de si mesmo foram utilizadas. Entretanto, não é possível confirmar que tais informações fornecidas estejam corretas. O entendimento dos participantes sobre as questões dos formulários é diretamente influenciado pela forma como as questões foram elaboradas; se a questão tiver sido mal formulada, o estudo pode ser afetado negativamente (WOHLIN *et al.*, 1999). No entanto, as respostas podem levar a contribuições sob diferentes óticas, o que é interessante para ampliar a aplicabilidade da abordagem.

Com suas limitações, esta avaliação não pode ser generalizada para outros grupos, porém foi útil para apontar deficiências da abordagem VisAr3D-Dynamic e oportunidades de melhoria. Como o estudo foi aplicado no contexto de manutenção de software, não se pode estender o mesmo resultado para outros domínios, como por exemplo para o ensino-aprendizagem. No entanto, como se espera uma oportunidade de aplicação da abordagem de análise dinâmica por meio de Realidade Virtual, considera-se importante uma avaliação semelhante para o domínio de ensino de modelagem de sistemas antes de uma adoção efetiva. Nesse sentido, pontos abordados neste estudo podem continuar sendo considerados, como experiência em orientação a objetos, porém outros podem ser modificados ou adicionados, de forma a incluir especificidades de cada área.

6.9. Considerações Finais

Neste capítulo, foi descrito o estudo desenvolvido para avaliar preliminarmente a abordagem VisAr3D-Dynamic. Neste estudo, cada participante executou os mesmos tipos de tarefas, porém com valores diferentes para cada ferramenta. Ao total foram 10 participantes distribuídos em dois grupos. Cada grupo tinha um arranjo de qual ferramenta seria apresentada primeiro para iniciar a execução das tarefas. Neste caso, 5 participantes do Grupo A executavam 5 tarefas na EA e mais 5 no protótipo. No Grupo B, 5 participantes executavam 5 tarefas no protótipo e mais 5 na ferramenta EA.

Além das análises realizadas nas seções anteriores, ao final do questionário de avaliação, foi perguntado sobre as vantagens e desvantagens da abordagem VisAr3D-Dynamic para cada participante. A Tabela 6.10 e Tabela 6.11 mostram os comentários dos participantes sobre as vantagens e desvantagens da abordagem, respectivamente.

Tabela 6.10 – Vantagens da abordagem VisAr3D-Dynamic segundo cada participante

“Principalmente execução passo a passo, link entre modelos e código”;
“A visualização de várias perspectivas, a interação entre elas, a observação rápida do comportamento dinâmico”;
“Engaja o usuário, além de possuir trechos de código. Também permite acompanhar a execução e controlá-la. Possibilita a visualização de métricas e navegação e exploração e navegação por camadas”;
“A disponibilidade e visualização de várias camadas de informações, as relações e o passo a passo da execução”;
“Permite ver a sequência de execução de forma dinâmica”;
“O principal aspecto positivo que achei foi a interligação entre os diferentes diagramas e a possibilidade de navegação entre eles e o código fonte de forma dinâmica e interativa e na mesma tela. Gostei bastante disto e nunca vi em nenhuma outra ferramenta”;
“Facilita muito a compreensão e navegação dos conceitos envolvidos, sobretudo para quem pode estar há um tempo sem mexer com o diagrama de sequência. Reunir vários diagramas juntos em um ambiente com linhas que os relacionam (eixo z) ajuda na interpretação”;
“Facilita muito a percepção da relação entre os artefatos, permite o acesso a todas as informações ao mesmo tempo”;
“Link e fluxo de execução são as principais qualidades do protótipo”;
“A interação do botão do mouse para alternar entre os contextos ajuda bastante, permitindo uma fácil navegação entre os modelos e o código”;

Tabela 6.11 – Desvantagens da abordagem VisAr3D-Dynamic segundo cada participante

“Em alguns momentos tive dificuldade em encontrar o melhor ângulo de visualização do modelo”;
“Muita informação disponibilizada de uma vez”;
“Navegação e aquelas linhas amarelas excessivas”;
“Dependendo do ângulo fica um pouco confuso, pois as camadas de trás estão com cores fortes, a navegação de voltar para as camadas anteriores ficou um pouco confusa”;
“O *Stop* reinicia tudo, seria bom parar onde está. O conteúdo do código não é sempre visível”;
“Ao final da execução, fiquei um pouco perdida em meio a tanta informação em uma única tela. Talvez incluir alguns filtros para deixar que o usuário defina qual o diagrama e informações a serem mostradas possa minimizar isto. Poderia também incluir um botão de 'localizar' para buscar termos textuais nos diagramas apresentados”;
“A função de *zoom* poderia ser melhor configurada usando CTRL e *scroll*”;
“O controle do *zoom* é um pouco difícil e dependendo da visão, algumas coisas não podem ser vistas ou sua visualização é prejudicada”;
“O *scroll* foi o principal problema que eu tive com a navegabilidade do protótipo. Pressionar não é tão intuitivo assim”;
“Acredito que a navegação quanto ao *zoom* é um pouco difícil, algumas vezes ele não funcionou durante o experimento”.

Um sucesso relativo pode ser considerado após a análise dos dados e comentários sobre as vantagens da abordagem. Grande parte dos participantes evidenciaram o uso da visualização do comportamento dinâmico executada passo a

passo; a organização das diferentes perspectivas de software no ambiente virtual e tridimensional e a possibilidade de percepção do *link* entre os diferentes diagramas e código-fonte como pontos fortes da abordagem. No entanto, como oportunidades de melhorias, a forma como foi implementado o *zoom* com o *mouse* foi motivo de desconforto e problema com o protótipo. Além disso, mecanismo de filtragem, tal como busca por elementos textuais, podem melhorar na busca por informações complexas.

CAPÍTULO 7 - CONCLUSÕES

7.1. Epílogo

A análise dinâmica é uma técnica tipicamente adotada para compreensão de software durante o processo de manutenção e evolução, pois fornece uma "imagem precisa" devido à captura de informações do comportamento corrente do sistema, conhecidas como rotas de execução. Estas rotas registram informações sobre a ordem temporal da troca de mensagens entre objetos, de acordo com um cenário ao qual se deseja investigar (DIT *et al.* 2013). No entanto, visualizar informação dinâmica emerge como um problema de escalabilidade, devido ao grande volume de dados, podendo gerar sobrecarga cognitiva durante o processo de manutenção (CORNELISSEN *et al.* 2011).

Para lidar com esta questão de escalabilidade foram propostas abordagens para a redução de rotas de execução e metáforas visuais não tradicionais. No entanto, técnicas de redução podem omitir informações importantes durante o processo de compreensão, enquanto que visualizações não tradicionais podem adicionar mais sobrecarga cognitiva devido à dificuldade de relacionar as formas visuais com os dados da execução do software (CORNELISSEN *et al.* 2011). Segundo MALETIC *et al.* (2002), novos dispositivos de visualização e interação, bem como novas representações, podem apoiar a compreensão de um grande volume de dados, tal como a utilização de Realidade Virtual (RV).

Este trabalho teve como principal objetivo apresentar a abordagem para apoiar a análise e compreensão do comportamento dinâmico de software, denominada VisAr3D-Dynamic. Visando o desenvolvimento da abordagem, a revisão da literatura identificou alguns conceitos importantes sobre Realidade Virtual (Capítulo 2) e análise dinâmica por meio de visualizações (Capítulo 3). Com o conhecimento reunido, foi possível verificar técnicas interessantes e necessárias para a construção da abordagem (apresentada no Capítulo 4) e sua implementação (Capítulo 5).

Um estudo de observação foi conduzido (Capítulo 6) com a finalidade de observar as contribuições da interação com modelos UML e código-fonte em um ambiente virtual e tridimensional. A partir dos resultados obtidos, verificou-se que há indícios de que a adoção da abordagem proposta possa trazer benefícios a seus usuários,

atuando como mecanismo de suporte e promovendo a compreensão do comportamento dinâmico. Algumas modificações foram indicadas para tornar seu uso mais intuitivo, adicionando novas funcionalidades e aprimorando as existentes.

7.2. Contribuições

Esta dissertação apresentou os resultados de um trabalho de pesquisa que visou propor uma abordagem para apoiar a compreensão do comportamento dinâmico de sistemas de larga escala orientados a objetos, além de iniciar o desenvolvimento de uma API para apoiar a renderização de modelos UML para aplicações no contexto de engenharia de software. Entre as principais contribuições deste trabalho, podemos destacar:

- Utilização da Realidade Virtual como técnica para apoiar a análise dinâmica de sistemas de software complexo. Além disso, considera-se como contribuição deste trabalho, a discussão sobre visualização 3D e sobre RV.
- O estudo e a comparação de diferentes abordagens sobre análise dinâmica guiada por visualização de software descritos na literatura, destacando os pontos positivos e negativos mais comuns destas abordagens;
- A definição de uma API capaz de ser reutilizada e expandida para a utilização de outros modelos UML ou até outros artefatos;
- A avaliação preliminar da abordagem, através de um estudo de observação, com alunos e ex-alunos de pós-graduação, analisando a execução de determinadas tarefas de manutenção de software carentes por mecanismos que apoiem o entendimento do comportamento dinâmico do software.

7.3. Limitações

Fazendo uma análise crítica da abordagem proposta e do protótipo implementado, é possível perceber as limitações deste trabalho. As principais são listadas a seguir:

- O protótipo possui alta dependência com a *engine* Unity 3D. Apesar do desenvolvimento da API apoiar a construção e renderização do protótipo, as visualizações em 3D estão todas acopladas fortemente ao Unity 3D.

- A disposição dos modelos UML e o código-fonte no ambiente virtual é o principal diferencial em relação às outras técnicas de análise dinâmica por meio de visualizações. Contudo, quando todos os elementos de todas as perspectivas são exibidos no ambiente, a forma como foram programados geram um certo tipo de desconforto ao explorar as informações do software. Além disso, as cores fortes, apesar de não terem significado, também atrapalham na análise. Talvez, como solução, fosse possível aplicar uma filtragem em transparência nos elementos, ou então, ter a possibilidade de controlar quais perspectivas visualizar.

- A avaliação da abordagem foi limitada, sendo restrita a apenas um estudo de observação com apenas 10 pessoas. Para que exista uma evidência significativa do real ganho no uso desse tipo de abordagem e se a mesma possui escalabilidade para tratar de projetos reais, estudos adicionais deverão ser elaborados.

7.4. Trabalhos Futuros

A partir do que foi proposto e implementado ao longo deste trabalho, é possível identificar possibilidades de melhoria e novas opções para se expandir a abordagem e o protótipo aqui apresentados. Entre esses possíveis trabalhos futuros, destacamos:

- Um estudo experimental com uma população relevante, *i.e.*, maior número de participantes com alto grau de experiência em manutenção de sistemas, para validar que tipo de informação pode ser considerado pertinente no contexto sobre compreensão do comportamento dinâmico de sistemas de software;

- Implementar mais interatividade no protótipo implementado traria mais recursos aos desenvolvedores. Por exemplo, seria interessante que fosse implementado um sistema de filtros, onde os usuários pudessem escolher visualizar apenas aqueles artefatos que atendem a algum tipo de critério;

- Avaliar a abordagem proposta no contexto de ensino-aprendizagem sobre modelagem UML de sistemas. Por exemplo, poderia ser conduzido outro estudo onde alunos da disciplina de modelagem utilizariam o protótipo a fim de apoiar a compreensão do comportamento dinâmico, principalmente o *link* entre dos elementos entre as diferentes perspectivas do software;

- Evoluir a abordagem de modo que outros artefatos, *e.g.*, outros diagramas UML ou processos de negócios em BPMN, na forma de novas perspectivas possam ser adicionados ao ambiente virtual. Além disso, recursos que pudessem realizar a comparação entre diferentes cenários de execução poderiam também ser acrescentados à abordagem, como mais um recurso para apoiar na compreensão de sistemas complexos orientados a objetos.
- Exploração da imersão com dispositivos imersivos, tais como óculos de realidade virtual para visualização em 360 graus e transmitir a sensação de que o usuário está imerso no ambiente visualizado. Do ponto de vista de interação, dispositivos por meio de gestos poderiam solucionar o problema com *mouse*, como apontado por alguns participantes do estudo.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDREWS, J.H., 1998, “Testing using log file analysis: tools, methods, and issues”. In: *Proceedings of the 13th IEEE International Conference on Automated Software Engineering,* pp. 157-166, Honolulu, EUA, Outubro.

BALL, T., 1999, “The concept of dynamic analysis”. In: *Proceedings of the 7th European Software Engineering Conference*, pp. 216-234, Toulouse, França, Setembro.

BASILI, V., CALDIERA, G. & ROMBACH, H., 1994, "Goal Question Metric Paradigm", Encyclopedia of Software Engineering, v. 1, n. edited by John J. Marciniak, John Wiley & Sons, pp. 528-532.

BENNETT, C., MYERS, D., STOREY, M. A., & GERMAN, D., 2007, “Working with’monster’traces: Building a scalable, usable, sequence viewer”. In: *Proceedings of the 3rd International Workshop on Program Comprehension Through Dynamic Analysis* (PCODA), pp. 1-5, Vancouver, Canadá.

BENNETT, C., MYERS, D., STOREY, M.A., GERMAN, D.M., OUELLET, D., SALOIS, M. & CHARLAND, P., 2008, “A survey and evaluation of tool features for understanding reverse-engineered sequence diagrams”. In: *Journal of Software Maintenance and Evolution: Research and Practice*, 20(4), pp.291-315.

BENNETT, K. H., & RAJLICH, V. T., 2000, “Software maintenance and evolution: a roadmap”. In: *Proceedings of the Conference on the Future of Software Engineering*, pp. 73-87, Limerick, Irlanda, Junho.

BIGGERSTAFF, T. J., MITBANDER, B. G., & WEBSTER, D., 1993, “The concept assignment problem in program understanding”. In: *Proceedings of the 15th International Conference on Software Engineering,* pp. 482-498, Baltimore, EUA, Maio.

BIOCCA, F. & LEVY, M.R., 2013, “Communication in the age of virtual reality”. Routledge.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I., 2005, “The Unified Modeling Language User Guide”. 2 ed. Addison-Wesley Professional.

BOWMAN, D., KRUIJFF, E., LAVIOLA JR, J.J. & POUPYREV, I., 2004, "3D User Interfaces: Theory and Practice". In: *CourseSmart e Textbook*. Addison-Wesley.

BREIVOLD, H. P., & LARSSON, M., 2007, "Component-based and service-oriented software engineering: Key concepts and principles". In: *33rd Conference on Software Engineering and Advanced Applications*, pp. 13-20, Washington, EUA, Agosto.

BROOKS, F.P., 1999, "What's real about virtual reality?". In: *IEEE Computer graphics and applications*, *19*(6), pp.16-27.

CASERTA, P. & ZENDRA, O., 2011, "Visualization of the static aspects of software: A survey". In: *IEEE transactions on visualization and computer graphics*, 17(7), pp.913-933.

CHEN, C., 2006, "Information visualization: Beyond the horizon". Springer Science & Business Media.

CHHABRA, J.K. & GUPTA, V., 2010, "A survey of dynamic software metrics". In: *Journal of computer science and technology*, 25(5), pp.1016-1029.

CORBI, T. A., 1989, "Program understanding: Challenge for the 1990s". IBM Systems Journal, 28(2), 294-306.

CORNELISSEN, B., ZAIDMAN, A., HOLTEN, D., MOONEN, L., VAN DEURSEN, A., & VAN WIJK, J. J., 2008, "Execution trace analysis through massive sequence and circular bundle views". In: *Journal of Systems and Software*, 81(12), 2252-2268.

CORNELISSEN, B., ZAIDMAN, A., VAN DEURSEN, A., MOONEN, L. & KOSCHKE, R., 2009, "A systematic survey of program comprehension through dynamic analysis". In: *IEEE Transactions on Software Engineering*, *35*(5), pp.684-702.

CORNELISSEN, S.G.M., 2009, "Evaluating Dynamic Analysis Techniques for Program Comprehension", PhD. Thesis, Delft University of Technology.

CRAIG, R.D. & JASKIEL, S.P., 2002, "Systematic software testing". Artech House.

CRUZ-NEIRA, C., SANDIN, D.J., DEFANTI, T.A., KENYON, R.V. & HART, J.C., 1992. “The CAVE: audio visual experience automatic virtual environment”. In: *Communications of the ACM*, *35*(6), pp.64-73.

DAVISON, A., 2005, “Killer game programming in Java”. O'Reilly Media, Inc.".

DE ALMEIDA, A. S. & LARA, A.Q., 1999. “O potencial de aplicação de sistemas de visualização e realidade virtual na atividade upstream de petróleo”. In: *Congresso Brasileiro de Engenharia Mecânica (COBEM’99), pp. 1-8, Águas de Lindóias, São Paulo, Brasil, Novembro.*

DE PAUW, W., LORENZ, D., VLISSIDES, J., WEGMAN, M., 1998, “Execution Patterns in Object-Oriented Visualization”. In: *Proceedings of the Conference on Object-Oriented Technologies and Systems*, USENIX, pp. 219–234, San Antonio, EUA, Fevereiro.

DIEHL, S., 2007, "Software visualization: visualizing the structure, behaviour, and evolution of software". Springer Science & Business Media.

DIONISIO, J.D.N. & GILBERT, R., 2013, “3D Virtual worlds and the metaverse: Current status and future possibilities”. In: *ACM Computing Surveys (CSUR)*,*45*(3), p.34.

DIT, B., REVELLE, M., GETHERS, M. & POSHYVANYK, D., 2013, “Feature location in source code: a taxonomy and survey”. In: *Journal of software: Evolution and Process*, 25(1), pp.53-95.

DUGERDIL, P.H. & ALAM, S., 2008, “Execution trace visualization in a 3D space”. *In: Proceeding of the 5th IEEE International Conference on Information Technology : New Generations (ITNG 2008), pp. 38-43, Las Vegas, Apr.*

ELLIOTT, A., PEIRIS, B. & PARNIN, C., 2015, “Virtual reality in software engineering: Affordances, applications, and challenges”. In: *Proceedings of the 37th International Conference Software Engineering* (ICSE'15), Vol. 2, pp. 547-550. Florence, Itália, Maio.

FERNANDES, F. A.; RODRIGUES, C. S. & WERNER, C. M., 2015, “Um Ambiente de Realidade Virtual para Apoiar a Compreensão de Aspectos Dinâmicos do Software”. In: *Anais do II Fórum de Educação em Engenharia de Computação*

V Simpósio Brasileiro sobre Engenharia de Sistemas de Computação, pp. 1-4, Foz do Iguaçu, Brasil, Novembro.

FERNANDES, F. A.; RODRIGUES, C. S. & WERNER, C. M., 2016, "Avaliação Heurística de um Ambiente Virtual para Análise de Rotas de Execução de Software". In: *Anais do VII Congresso Brasileiro de Software: Teoria e Prática (CBSoft) – IV Workshop sobre Visualização, Evolução e Manutenção (VEM)*, pp. 41-48, Maringá, Brasil, Setembro.

FITTKAU, F., WALLER, J., WULF, C. & HASSELBRING, W., 2013, "Live trace visualization for comprehending large software landscapes: The ExplorViz approach". *In: Software Visualization (VISSOFT)*. First IEEE Working Conference on (pp. 1-4). IEEE.

FJELDSTAD, R. K., & HAMLEN, W. T., 1983, "Application program maintenance study: Report to our respondents". Tutorial on Software Maintenance, IEEE Computer Society Press, pp. 13 - 30, Los Alamitos, EUA, Abril.

FOWLER, M., 2014, "UML Essencial: um breve guia para linguagem-padrão de modelagem de objetos". 3 ed. Porto Alegre. Bookman.

FRANCK, G., SARDESAI, M., WARE, C., 1995, "Layout and structuring object oriented software in three dimensions", In: *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research (CASCON '95)*, pp. 22-22, Toronto, Ontario, Canadá, Novembro.

GERSHON, N. D., 1992, "From perception to visualization". In: *Computer graphics*, 26(2), pp.414-415.

GOOGLE, 2016, Google Cardboard. Disponível em: https://vr.google.com/cardboard. Acesso em: 7 de outubro de 2016.

GREEVY, O., LANZA, M. & WYSSEIER, C., 2006, "Visualizing live software systems in 3D". *In: Proceedings of the 2006 ACM symposium on Software visualization,* pp. 47-56, Nova Iorque, EUA, Setembro.

HTC, 2016, HTC Vive. Disponível em: <https://www.vive.com/us/>. Acesso em: 7 de outubro de 2016.

IRANI, P. & WARE, C., 2003, “Diagramming information structures using 3D perceptual primitives”. In: *ACM Transactions on Computer-Human Interaction (TOCHI)*, *10*(1), pp.1-19.

JERDING, D. F.; STASKO, J. T, 1998, “The information mural: A technique for displaying and navigating large information spaces”. In: *IEEE Transactions on Visualization and Computer Graphics*, v. 4, n. 3, p. 257-271.

JURISTO, N. & MORENO, A.M., 2013, “Basics of software engineering experimentation”. Springer Science & Business Media.

KICZALES, G., HILSDALE, E., HUGUNIN, J., KERSTEN, M., PALM, J. & GRISWOLD, W.G., 2001, “An overview of AspectJ”. In: *Proceedings of the 15th European Conference on Object-Oriented Programming*, pp. 327-354, Londres, Inglaterra, Junho.

KIRNER, C., & TORI, R., 2004, “Introdução à realidade virtual, realidade misturada e hiper-realidade”. Realidade Virtual: Conceitos, Tecnologia e Tendências. 1ed. São Paulo, 1, 3-20.

KOENEMANN, J., & ROBERTSON, S. P., 1991, “Expert problem solving strategies for program comprehension”. In: *Proceedings of the Conference on Human Factors in Computing Systems*, pp. 125-130, New Orleans, EUA, Abril.

LEAP MOTION, 2016, Leap Motion. Disponível em: https://www.leapmotion.com/. Acesso em: 7 de outubro de 2016.

LEHMAN, M. M., & BELADY, L. A., 1985, “Program evolution: processes of software change”. Academic Press Professional, Inc.

LEHMAN, M. M., 1980, “Programs, life cycles, and laws of software evolution”. In: *Proceedings of the IEEE*, 68(9), 1060-1076.

LEHMAN, M. M., 1996, "Laws of Software Evolution Revisited", *Software Process Technology*, Springer Berlin / Heidelberg, pp. 108-124.

LETOVSKY, S., 1987, “Cognitive processes in program comprehension”. Journal of Systems and software, 7(4), 325-339.

LIENTZ, B. P., & SWANSON, E. B., 1980, “Software maintenance management”. Addison-Wesley Longman Publishing Co., Inc.

LIENTZ, B. P., SWANSON, E. B., & TOMPKINS, G. E., 1978, “Characteristics of application software maintenance”. *Communications of the ACM*, 21(6), 466-471.

MACKINLAY, J., 1986, “Automating the design of graphical presentations of relational information”. In: *Acm Transactions On Graphics (Tog)*, *5*(2), pp.110-141.

MALETIC, J. I., MARCUS, A., & COLLARD, M. L., 2002, “A task oriented view of software visualization”. In: *Proceeding of the 1st International Workshop on Visualizing Software for Understanding and Analysis* (*VISSOFT 2002),* pp. 32-40, Washington, EUA, Junho.

MALETIC, J.I., LEIGH, J., MARCUS, A. & DUNLAP, G., 2001, “Visualizing object-oriented software in virtual reality”. In: *Proceedigns of the 9th International Workshop on Program Comprehension (IWPC)*, pp. 26-35, Toronto, Canadá, Maio.

MATHIASSEN, L., MUNK-MADSEN, A., NIELSEN, P. A., & STAGE, J., 2000, “Object-oriented analysis & design”. Aalborg: Forlaget Marko.

MAYER, R.E & CHANDLER, P., 2001, "When learning is just a click away: Does simple user interaction foster deeper understanding of multimedia messages?" In: *Journal of educational psychology*, 93(2), 390.

MENS, T., WERMELINGER, M., DUCASSE, S., DEMEYER, S., HIRSCHFELD, R., & JAZAYERI, M., 2005, “Challenges in software evolution”. In: *Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE'05)*, pp. 13-22, Washington, EUA, Setembro.

MICROSOFT, 2016, Kinect for Xbox One. Disponível em: http://www.xbox.com/en-US/xbox-one/accessories/kinect. Acesso em: 7 de outubro de 2016.

MILI, R. & STEINER, R., 2002, “Software Engineering - Introduction”, Revised Lectures on Software Visualization, Int’l Seminar, pp. 129-137.

MUHANNA, M.A., 2015, “Virtual reality and the CAVE: Taxonomy, interaction challenges and research directions”. *Journal of King Saud University-Computer and Information Sciences*, *27*(3), pp.344-361.

MURUGESAN, S., DESHPANDE, Y., HANSEN, S. & GINIGE, A., 2001, “Web engineering: A new discipline for development of web-based systems”. *In:* Web Engineering (pp. 3-13). Springer Berlin Heidelberg.

NAKATSU, R. & TOSA, N., 2000, “Active immersion: the goal of communications with interactive agents”. In: *Proceedings of the 4th International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies,* Vol. 1, pp. 85-89, Brighton, Inglaterra, Agosto.

OCULUS, 2016, Oculus Rift. Disponível em: Acesso em: 7 de outubro de 2016.

OMG, 2007, *XMI 2.2.1 Specification*, XMI specification formal/2007-12-01, Object Management Group.

OSHIBA, T. & TANAKA, J., 1999, "3D-PP: three-dimensional visual programming system”. In: *Proceedings of the 1999 IEEE Workshop on Visual Languages,* pp. 189-190.

PACIONE, M.J., ROPER, M. & WOOD, M., 2003, “A comparative evaluation of dynamic visualisation tools”. In: *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE '03)*, pp. 80-89, Washington, DC, USA, Novembro.

PARNAS, D. L., 1994, “Software aging”. In: *Proceedings of the 16th International Conference on Software Engineering* (pp. 279-287). IEEE Computer Society Press.

PARSONS, S. & COBB, S., 2011, “State-of-the-art of virtual reality technologies for children on the autism spectrum”. In: *European Journal of Special Needs Education*, *26*(3), pp.355-366.

PIGOSKI, T. M., 1996, “Practical software maintenance: best practices for managing your software investment”. Wiley Publishing.

PIMENTEL, K. & TEIXEIRA, K., 1993, “Virtual reality through the new looking glass”. Windcrest/McGraw-Hill, Blue Ridge Summit, PA.

PLAYSTATION, 2016, PlayStation VR. Disponível em: https://www.playstation.com/en-us/explore/playstation-vr/. Acessado em: 7 de outubro de 2016.

PREDDY, S.M. & NANCE, R.E., 2002, "Key requirements for CAVE simulations". In: *Proceedings of the 2002 Winter Simulation Conference (WSC)*, Vol. 1, pp. 127-135, San Diego, EUA, Dezembro.

PRICE, B. A., BAECKER, R. M., & SMALL, I. S., 1993, "A principled taxonomy of software visualization". Journal of Visual Languages & Computing, 4(3), 211-266.

RADFELDER, O. & GOGOLLA, M., 2000, "On better understanding UML diagrams through interactive three-dimensional visualization and animation". In: *Proceedings of the working conference on Advanced visual interfaces* (pp. 292-295). Palermo, Itália, Maio.

RODRIGUES, C.S.C., 2012, "VisAr3D-Uma Abordagem Baseada em Tecnologias Emergentes 3D para o Apoio à Compreensão de Modelos UML". Tese de D.Sc., Universidade Federal do Rio de Janeiro, COPPE, Rio de Janeiro, Brasil. Disponível em: http://reuse.cos.ufrj.br/files/publicacoes/doutorado/Dou_ClaudiaSusie.pdf

ROMAN, G. C., & COX, K. C., 1993, "A taxonomy of program visualization systems". IEEE Computer, 26, n. 12, Dezembro 1993. 11-24.

SAMSUNG, 2016, Samsung Gear VR. Disponível em: <http://www.samsung.com/global/galaxy/gear-vr/>. Acesso em: 7 de outubro de 2016.

SKYPE, 2017, Skype. Disponível em: https://www.skype.com/en/. Acesso em: 26 de Abril de 2017.

SHERMAN, W.R. & CRAIG, A.B., 2002, "Understanding virtual reality: Interface, application, and design". Elsevier.

SHNEIDERMAN, B., 1996, "The eyes have it: A task by data type taxonomy for information visualizations". In: *Visual Languages*, 1996. Proceedings., IEEE Symposium on (pp. 336-343). IEEE.

SHULL, F., CARVER, J. & TRAVASSOS, G.H., 2001, “An empirical methodology for introducing software processes”. *In:* ACM SIGSOFT Software Engineering Notes (Vol. 26, No. 5, pp. 288-296). ACM.

SOMMERVILLE, I., 2011, “Engenharia de Software”. 9 ed. São Paulo: Pearson Prentice Hall.

SOMMERVILLE, I., CLIFF, D., CALINESCU, R., KEEN, J., KELLY, T., KWIATKOWSKA, M., & PAIGE, R., 2012, "Large-scale complex IT systems". In: *Communications of the ACM*, 55(7), 71-77.

SYSTÄ, T., KOSKIMIES, K., & MÜLLER, H, 2001, “Shimba - an environment for reverse engineering Java software systems”. In: *Software: Practice and Experience*, 31(4), 371-394.

TEAMVIEWER, 2017, TeamViewer. Disponível em: https://www.teamviewer.com/en/. Acesso em: 26 de Abril de 2017.

TEYSEYRE, A.R. & CAMPO, M.R., 2009, “An overview of 3D software visualization”. In: *Journal IEEE Transactions on Visualization and Computer Graphics*, v. 15(1), pp.87-105.

TORI, R. & KIRNER, C., 2006, “Fundamentos de Realidade Virtual”. In: *Fundamentos e Tecnologia de Realidade Virtual e Aumentada*, v. 1, Editora SBC - Sociedade Brasileira de Computação, Porto Alegre, pp. 22-38.

TORY, M., KIRKPATRICK, A.E., ATKINS, M.S. & MOLLER, T., 2006, “Visualization task performance with 2D, 3D, and combination displays”. In: *IEEE transactions on visualization and computer graphics*, *12*(1), pp.2-13.

TRAVASSOS, G.H., GUROV, D. & AMARAL, E.A.G.G., 2002, “Introdução à engenharia de software experimental”. Relatório Técnico ES 590/02, COPPE/UFRJ. Disponível em: http://www.cos.ufrj.br/uploadfile/es59002.pdf.

TULACH, J., 2008, “Practical API design: Confessions of a java framework architect”. Apress.VAN DAM, A., FORSBERG, A. S., LAIDLAW, D. H., LAVIOLA, J. J., & SIMPSON, R. M., 2000, “Immersive VR for scientific visualization: A progress report”. IEEE Computer Graphics and Applications, 20(6), 26-52.

VON MAYRHAUSER, A., & VANS, A. M., 1995, “Program comprehension during software maintenance and evolution”. Computer, 28(8), 44-55.

WALKER, R. J., MURPHY, G. C., FREEMAN-BENSON, B., WRIGHT, D., SWANSON, D. & ISAAK, J., 1998, “Visualizing dynamic software system information through high-level models”. In: *ACM SIGPLAN Notices*. Vol. 33, No. 10, pp. 271-283.

WARE, C., HUI, D. & FRANCK, G., 1993, “Visualizing object oriented software in three dimensions”, In: *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research (CASCON '93)*, pp. 612-620, Toronto, Canadá, Outubro.

WEBSTER, R.W., ZIMMERMAN, D.I., MOHLER, B.J., MELKONIAN, M.G. AND HALUCK, R.S., 2001, “A prototype haptic suturing simulator. Studies in health technology and informatics”. In: *Proceedings of the Medicine Meets Virtual Reality*, pp.567-569, San Diego, EUA, Janeiro.

WEN, J., 1995, “Exploiting orthogonality in three dimensional graphics for visualizing abstract data”. In: *Technical Report CS-95-20, Dept. of Computer Science*, Brown University.

WERNER, C., MATTOSO, M., BRAGA, R. *et al.*, 1999, “Odyssey: Infraestrutura de Reutilização baseada em Modelos de Domínio”, In: *Caderno de Ferramentas do XIII Simpósio Brasileiro de Engenharia de Software (XIII SBES)*, pp.17-20, Florianópolis, Brasil, Outubro.

WOHLIN, C.; RUNESON, P.; HÖST, M., 1999, “Experimentation in Software Engineering: An Introduction”. 1st ed. Springer.

YACOUB, S. M., AMMAR, H. H., & ROBINSON, T. 2000, "A methodology for architectural-level risk assessment using dynamic metrics". In: *Software Reliability Engineering*, 2000. ISSRE 2000. Proceedings. 11th International Symposium on (pp. 210-221). IEEE.

ZAIDMAN, A., 2006, “Scalability solutions for program comprehension through dynamic analysis”. Ph.D. Thesis, Universiteit Antwerpen. Faculteit Wetenschappen.

ZAYOUR, I. & LETHBRIDGE, T.C., 2001, Adoption of reverse engineering tools: a cognitive perspective and methodology. In: *Proceedings of the 9th International Workshop on Program Comprehension (IWPC)*, pp. 245-255, Toronto, Canadá, Maio.

ZHAO, Z.X., 2002, "Virtual reality technology: an overview". In: *Journal of Southeast University*, *32*(2A), pp.1-10.

ZHAO, C., ZHANG, K., HAO, J. & WONG, W.E., 2009, "Visualizing multiple program executions to assist behavior verification". *In: Proceedings of the 3rd IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, pp. 113-122, Shanghai, China, July.

APÊNDICE A - FORMULÁRIO DE CONSENTIMENTO

VISAR3D-DYNAMIC

Este estudo tem como objetivo avaliar a abordagem VisAr3D-Dynamic, considerando seu apoio à compreensão do comportamento dinâmico de software por meio de diagramas UML em um ambiente virtual tridimensional. Para isso, será utilizado um protótipo, que implementa as características da abordagem.

IDADE

Eu declaro ter mais de 18 anos de idade e concordar em participar de um estudo experimental conduzido por Filipe Arantes Fernandes na COPPE/UFRJ.

PROCEDIMENTO

Este estudo ocorrerá em uma única sessão, que incluirá um treinamento sobre a abordagem VisAr3D-Dynamic. Eu entendo que, uma vez o experimento tenha terminado, os trabalhos que desenvolvi serão estudados visando entender a eficiência dos procedimentos e as técnicas que me foram ensinadas.

CONFIDENCIALIDADE

Toda informação coletada neste estudo é confidencial, e meu nome não será divulgado. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

BENEFÍCIOS, LIBERDADE DE DESISTÊNCIA

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e ensinado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

PESQUISADOR RESPONSÁVEL

Filipe Arantes Fernandes
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

PROFESSORES RESPONSÁVEIS

Profª. Claudia Susie C. Rodrigues

Profª. Cláudia M.L. Werner

Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Nome (em letra de forma):_____

Assinatura: _____ **Data:** _____

APÊNDICE B - CARACTERIZAÇÃO DO PARTICIPANTE

Código do Participante: _____ Data: _____

Esta fase da pesquisa tem como objetivo obter informações sobre a sua experiência acadêmica e profissional.

Por favor, NÃO inclua nenhum detalhe que poderá identificá-lo.

Perfil do participante

1) Formação Acadêmica:

( ) Doutorado
( ) Doutorado em Andamento
( ) Mestrado
( ) Mestrado em Andamento
( ) Graduação
( ) Graduação em Andamento
( ) Outro: _____

Ano de ingresso: _____ Ano de conclusão (de previsão): _____

2) Em qual contexto utiliza/utilizou diagramas UML?

( ) Como professor/instrutor/monitor
( ) Como aluno
( ) Como profissional da indústria
( ) Outro:_____

3) Qual é sua experiência com modelagem UML? (se necessário, marque mais de um item)

( ) Já li material sobre modelagem UML
( ) Já participei de um curso sobre modelagem UML
( ) Nunca fiz uma modelagem UML
( ) Já fiz modelagem UML para uso próprio
( ) Já fiz modelagem UML como parte de uma equipe, relacionada a um curso
( ) Já fiz modelagem UML como parte de uma equipe na indústria
Outro:_____

4) Qual é sua experiência em compreender a troca de mensagens entre objetos de um software orientado a objeto? (se necessário, marque mais de um item)

( ) Já li material sobre este assunto
( ) Já participei de um curso sobre este assunto
( ) Nunca precisei entender sobre este assunto
( ) Já precisei compreender a troca de mensagens entre objetos para uso próprio
( ) Já precisei compreender a troca de mensagens entre objetos como parte de uma equipe relacionada a um curso.
( ) Já precisei compreender a troca de mensagens entre objetos como parte de uma equipe na indústria.
( ) Outro:_____

5) Inclua o número de semestres ou número de anos de experiência relevante em **modelagem** UML. Por favor, explique sua resposta. (Ex: "Eu trabalhei por 2 anos fazendo modelagem UML na indústria")

_____

_____

_____

6) Inclua o número de semestres ou número de anos de experiência relevante em **programação**. Por favor, explique sua resposta. (Ex: "Eu trabalhei por 2 anos como programador Java na indústria")

_____

_____

_____

Marque uma opção, indicando o grau de sua experiência utilizando a escala dos 5 pontos abaixo:
0 = nenhum
1 = estudei em aula ou em livro
2 = pratiquei em projetos em sala de aula
3 = usei em projetos pessoais
4 = usei em projetos na indústria

7) Orientação a Objetos: (0) (1) (2) (3) (4)

8) Padrões de Projeto: (0) (1) (2) (3) (4)

9) Java: (0) (1) (2) (3) (4)

10) Outra linguagem: (0) (1) (2) (3) (4)

11) Qual ferramenta UML utiliza/utilizou para modelar diagramas de sequência UML?

_____

12) Qual ferramenta utiliza/utilizou para apoiar a compreensão da troca de mensagens entre objetos?

_____

Em relação ao maior sistema em que atuou

13) Qual a ferramenta de modelagem utilizada?

_____

14) Indique a quantidade aproximada de objetos (*lifelines*) do maior diagrama de sequência:

( ) até 20
( ) de 21 até 50
( ) de 51 até 100
( ) de 101 a 200
( ) mais de 200
( ) nunca utilizei

15) Qual a ferramenta para apoiar a compreensão da troca de mensagens entre objetos foi utilizada?

_____

16) Qual o tipo de representação dos dados utilizado pela ferramenta de apoio à compreensão da troca de mensagens entre objetos?

( ) Textual ( ) Diagramas UML
( ) Grafos ( ) Outras_____

17) Caso tenha utilizado engenharia de reversa, por favor, descreva o cenário brevemente.

_____

_____

_____

_____

_____

APÊNDICE C - TAREFAS

Código do Participante: _____ Data: _____

INSTRUÇÕES

Este estudo será acompanhado por meio de anotações feitas pelo pesquisador. Sempre que possível, verbalize seus pensamentos, para que o experimentador possa melhor avaliar os resultados obtidos. Pergunte e comente tudo que achar necessário.

CONTEXTUALIZAÇÃO

Você está participando deste estudo como programador e utilizará o ambiente virtual para análise de rotas de execução para responder algumas questões relacionadas à visualização de um diagrama de sequência.

Para responder as perguntas de 1 a 6, utilize a ferramenta EA (*Enterprise Architect*)

1) Identifique 2 classes com alto acoplamento dinâmico.

_____

_____

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

2) Identifique uma classe no pacote *feature* com um forte acoplamento para o pacote *generic*.

_____

_____

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

3) Identifique a primeira classe que instancia a classe *LigAssociacao*.

_____

_____

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

4) Localize a *lifeline* representada pela classe *ItemModelo* e liste todos os métodos que podem ser executados a partir dela.

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

5) Listar os pacotes utilizados até a execução da *lifeline FabricaSemanticos*.

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

6) Quais classes pertencem ao pacote *model*?

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

Para responder as perguntas de 7 a 12, utilize o protótipo VisAr3D-Dynamic

7) Identifique a classe que possui a maior quantidade de chamadas de métodos.

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

8) Identifique uma classe no pacote *model* que possui muitos relacionamentos para o pacote *feature*.

_____

_____

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

9) Identifique a classe que faz a primeira chamada de método à classe *FabricaSemanticos*.

_____

_____

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

10) Liste todos os métodos da classe *NoFeatureDominio* que podem ser executados por ela.

_____

_____

_____

_____

_____

_____

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

11) Quais os pacotes utilizados durante a execução da *lifeline LigAssociacao* até a *NoPacote*?

_____

_____

_____

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

12) O pacote *feature* possui quais classes?

_____

_____

_____

_____

Esta tarefa foi ( ) fácil ( ) média ( ) difícil

APÊNDICE D - QUESTIONÁRIO DE AVALIAÇÃO

Código do Participante: _____ Data: _____

REALIZAÇÃO DA TAREFA

1) Você conseguiu efetivamente realizar todas as tarefas propostas? Comente, se necessário.

( ) Sim ( ) Não ( ) Parcialmente

_____

_____

_____

2) Você ficou satisfeito com o resultado final das tarefas? Comente, se necessário.

( ) Sim ( ) Não ( ) Parcialmente

_____

_____

_____

TREINAMENTO

3) Você considera que o treinamento aplicado para o uso das ferramentas (protótipo e EA) e para a realização das tarefas foi suficiente? O que poderia ser acrescentado/modificado?

( ) Sim ( ) Não ( ) Parcialmente

_____

_____

_____

4) Você considera que existiu alguma dificuldade na interpretação das informações apresentadas sobre as ferramentas (protótipo e EA)? Se sim, por favor, explique.

( ) Sim ( ) Não ( ) Parcialmente

_____

_____

_____

COMPARAÇÃO ENTRE VISUALIZAÇÃO 2D E VISUALIZAÇÃO 3D

A seguir serão apresentados alguns tópicos que servem para comparar a visualização 2D com a visualização 3D. Escolha o tipo de visualização que você considera que **melhor contribuiu**. Em seguida, faça um comentário a respeito, se necessário.

5) Apoio à compreensão da rota de execução através do diagrama de sequência UML?

( ) 2D ( ) 3D ( ) Não se aplica

6) Apoio à exploração dos modelos?

( ) 2D ( ) 3D ( ) Não se aplica

7) Apoio à resolução das tarefas?

( ) 2D ( ) 3D ( ) Não se aplica

8) Redução da complexidade da visualização?

( ) 2D ( ) 3D ( ) Não se aplica

9) Disponibilização de ambiente mais intuitivo?

( ) 2D ( ) 3D ( ) Não se aplica

10) Facilidade na exploração de informações entre digramas e código?

( ) 2D ( ) 3D ( ) Não se aplica

11) Facilidade na exploração de informações a cada mensagem executada?

( ) 2D ( ) 3D ( ) Não se aplica

12) Facilidade de obtenção de informações sobre aspectos de qualidade de software?

( ) 2D ( ) 3D ( ) Não se aplica

13) Facilidade no entendimento sobre herança e polimorfismo quando o software é executado?

( ) 2D ( ) 3D ( ) Não se aplica

14) Suporte à compreensão do comportamento dinâmico?

( ) 2D ( ) 3D ( ) Não se aplica

15) Melhor percepção de informações relacionadas entre os modelos e o código?

( ) 2D ( ) 3D ( ) Não se aplica

16) Melhor engajamento na exploração de informações acerca do comportamento dinâmico?

( ) 2D ( ) 3D ( ) Não se aplica

USO DO PROTÓTIPO VISAR3D-DYNAMIC

17) Indique um grau de relevância para cada função do protótipo VisAr3D-Dynamic. Marque 1 para nenhuma e 5 para máxima relevância.

Funções do protótipo	**Escala**
Camera Reset	(1) (2) (3) (4) (5)
Relationship between layers	(1) (2) (3) (4) (5)
Dynamic Coupling	(1) (2) (3) (4) (5)
Play	(1) (2) (3) (4) (5)
Stop	(1) (2) (3) (4) (5)
Rewind	(1) (2) (3) (4) (5)
Forward	(1) (2) (3) (4) (5)
As teclas A, S, D e W	(1) (2) (3) (4) (5)
As teclas ← → ↑ ↓	(1) (2) (3) (4) (5)
O botão esquerdo do mouse para movimentação da câmera	(1) (2) (3) (4) (5)
O botão do meio do mouse para zoom	(1) (2) (3) (4) (5)
O botão direito do mouse para abrir o menu de contexto	(1) (2) (3) (4) (5)
Slider (botão deslizante para esquerda e direita)	(1) (2) (3) (4) (5)
Go to the Package	(1) (2) (3) (4) (5)
Go to the Class	(1) (2) (3) (4) (5)
Go to the Lifeline	(1) (2) (3) (4) (5)
Go to the Message in Lifeline	(1) (2) (3) (4) (5)
Go to the Code	(1) (2) (3) (4) (5)
Go to the Message in Code	(1) (2) (3) (4) (5)
Go to the Next Breakpoint	(1) (2) (3) (4) (5)

18) Liste os aspectos positivos do protótipo VisAr3D-Dynamic.

_____

_____

_____

19) Liste os aspectos negativos do protótipo VisAr3D-Dynamic.

_____

_____

_____

20) Por favor, adicione qualquer outro comentário desejado aqui.

_____

_____

Obrigado por sua colaboração!