Análise Comparativa de um Protótipo de IDE para Desenvolvimento de SMA Embarcados

Elaine Maria Pereira Siqueira, Gabriel Ramos, Thácito Raboni, Carlos Eduardo Pantoja, Nilson Mori Lazarin

Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
Rio de Janeiro – RJ – Brazil

{elaine.siqueira, gabriel.ramos, thacito.medeiros}@aluno.cefet-rj.br

{nilson.lazarin, carlos.pantoja}@cefet-rj.br

Abstract. *Nowadays, using Integrated Development Environments (IDEs) in creating software processes is widely adopted and has become a necessity to increase the time optimization in the development process. However, when developing embedded multi-agent systems, the tools for automating the development process still require improvement. So, this paper evaluates a specific IDE for the development of cognitive hardware supported by robotic agents, the chonIDE. It analyzed its graphical characteristics, features, and benefits as a tool to facilitate Embedded MAS programming. Furthermore, this paper proposes a few changes in the chonIDE layout to improve the embedded MAS designer experience.*

Resumo. *Usar um Ambiente de Desenvolvimento Integrado (IDE) como ferramentas auxiliar na criação de softwares é uma necessidade, principalmente num contexto em que a otimização de tempo e trabalho são características valiosas no progresso do mercado tecnológico. Entretanto, as ferramentas de desenvolvimento de sistemas multiagentes (SMA) embarcados atravessam etapas que ainda carecem de atualização neste sentido, dessa forma, este trabalho apresenta uma avaliação da chonIDE. Foram analisadas suas características gráficas, funcionalidades e benefícios como ferramenta para facilitar a programação de SMA Embarcados. Além disso, são apresentadas sugestões de alteração no layout da ferramenta para melhorar a experiência do desenvolvedor.*

1. Introdução

O Ambiente de Desenvolvimento Integrado (IDE) fornece muitas ferramentas que suportam o processo de desenvolvimento de software, incluindo editores para escrever e editar programas e depuradores para localizar erros de lógica em programas [Deitel and Deitel 2009]. Antes da existência das IDEs, os códigos fonte eram escritos em editores de texto comuns, como o bloco de notas do celular, por exemplo. Em seguida, o código era conduzido a um compilador, responsável por traduzi-lo de linguagem de alto nível (usada pelos programadores) para linguagem simbólica, isto é, de baixo

nível (usada pelas máquinas), para ser lido pelos processadores do computador, e as respostas aos erros, caso houvesse algum, eram anotadas pelo programador, para que este pudesse corrigi-los, usando novamente o editor de texto. Caso não houvesse erros, o código era levado para execução, e, se ele apresentasse um comportamento diferente do esperado pelo programador, era direcionado a um depurador. Este, por sua vez, auxilia na identificação dos problemas ocorridos, uma vez que é utilizado para realizar testes no software, a fim de encontrar quaisquer problemas ou bugs no código. Por fim, o código volta ao editor de texto para ser aperfeiçoado [Hou and Wang 2009].

Sendo assim, é possível verificar que a criação de uma aplicação antes do surgimento das IDEs era complicado, haja vista que todo o procedimento previamente detalhado se repetia sempre que surgisse algum erro, o que, por consequência, diminuía a eficiência no processo de desenvolvimento de um software, enquanto aumentava as demandas de tempo e trabalho para integrar e gerenciar todas as funcionalidades citadas. O aparecimento das IDEs, tal qual Eclipse, VSCode, IntelliJ, entre outras, trouxe atualizações, como debug, instrumental integrado em single-screen, integração com versionadores de código, etc, que contribuíram para resolver alguns dos problemas já abordados. Esta inovação, entretanto, carece de equipagem específica para incorporar soluções de Inteligência Artificial (IA) em SMA, dada a natureza distribuída destes. O ambiente de desenvolvimento explorado neste trabalho, a chonIDE [Souza de Jesus et al. 2022, Souza de Jesus et al. 2023], por outro lado, resolve essa questão, mas sofre da ausência de ferramental que auxilie o projetista no desenrolar da aplicação.

O objetivo desse artigo é realizar um comparativo que permita analisar as principais funcionalidades da chonIDE, e outras IDEs atuais, para propôr avanços no contexto de programação de SMA Embarcados. Com esse intuito, serão retomadas definições que contribuem para o entendimento do cenário em que esse novo software atua e destacada como ele pode colaborar para melhorar tais circunstâncias. Além disso, foram identificados duas funcionalidades para serem desenvolvidas: o versionamento de código e o debug integrado.

Este trabalho está estruturado da seguinte forma: na Seção 2 são apresentados conceitos que enriquecem a compreensão do cenário abordado; na Seção 3 são expostas características que dizem respeito a identidade da chonIDE; na Seção 4 exibe tabelas que contribuem visualmente para a análise das limitações e diferenciais da chonIDE em contraste com as outras três IDEs mencionadas; a Seção 5 há uma explicação do motivo de considerar especificamente as IDEs Eclipse, VSCode e IntelliJ relevantes para a seção anterior; a Seção 6 evidencia um novo protótipo do software que é o foco deste documento e, finalmente, na Seção 7 são apresentadas as considerações finais e a indicação de trabalhos futuros.

2. Referencial Teórico

A essência dos sistemas computacionais autônomos é a autogestão, cuja intenção é libertar os administradores de sistema dos detalhes de funcionamento e manutenção desse sistema e fornecer aos usuários uma máquina que funciona com desempenho máximo 24 horas por dia, 7 dias por semana [Kephart and Chess 2003]. Um agente é um sistema reativo que exibe algum grau de autonomia, no sentido de que delegamos alguma tarefa a ele

e o próprio sistema determina a melhor forma de realizar essa tarefa [Bordini et al. 2007]. Esses agentes são denominados cognitivos, visto que, nos agentes, há uma representação explícita do ambiente e dos membros da sociedade. Eles podem raciocinar sobre as ações tomadas no passado e planejar as ações a serem tomadas no futuro [Sichman et al. 1992]. Para tanto, um agente pode: perceber seu ambiente ou comunicar-se com outros agentes.

Devido a habilidade social, que caracteriza os agentes, isto é, capacidade de realizar interações com outros agentes [Bordini et al. 2007], o mais comum no projeto de um software é que se trabalhe com mais de um agente em um mesmo ambiente. Os SMA são sistemas compostos por múltiplos agentes autônomos e proativos, com capacidade de tomada de decisão e comunicação com outros. Sendo assim, esses agentes podem cooperar entre si para alcançar um objetivo comum ao SMA [Wooldridge 2000], isto é, realizar tarefas complexas [Wooldridge 2009]. Apesar de autônomos, os agentes de um SMA eventualmente seguem normas e assumem papéis no grupo que estão contidos, permitindo alinhar seus objetivos individuais, os quais os delegamos, com os objetivos do sistema e, portanto, convergindo para o cumprimento de atividades complexas [Manoel et al. 2022].

Diferentemente dos computadores pessoais, de propósito geral e programados pelo usuário final, um sistema é classificado como embarcado quando este é dedicado a uma única tarefa e interage continuamente com o ambiente a sua volta por meio de sensores e atuadores [Ball 2002] que, por sua vez, compõem o dispositivo físico no qual o sistema está necessariamente encapsulado. Em outras palavras, um sistema embarcado é um sistema computacional embutido/dedicado ao hardware de um dispositivo e responsável pelo controle e monitoramento deste hardware, realização de processos e comunicações com outros dispositivos [Marwedel 2021]. Baseado em um microprocessador, projetado para controlar uma função ou uma gama de funções [Heath 2002], ele faz parte de um sistema ainda maior, para implementar alguns dos requerimentos deste sistema [IEEE 1990], por exemplo: em semáforos, aparelhos de ar-condicionado (controle da temperatura), impressoras, etc. As principais características de classificação de um sistema embarcado são a sua capacidade computacional e a sua independência de operação [Ball 2002], já que funciona de maneira exclusiva [Heath 2002], o que o torna destaque nestes sentidos.

O produto da união de um SMA, devido as suas qualidades de autonomia, proatividade e capacidade deliberativa, ao hardware de um *high-end IoT device* [Ojo et al. 2018] (um computador de placa única como a Raspberry Pi, por exemplo), por conta de suas capacidade computacional e independência de operação, denomina-se SMA Embarcado. Já existe, na literatura, uma arquitetura consolidada sobre esse tipo de sistema, que se secciona em: raciocínio, interfaceamento, firmware e hardware [Pantoja et al. 2016].

A camada de raciocínio, hospedada em um dispositivo dedicado [Pantoja et al. 2016, Souza de Castro et al. 2022], é a camada onde está situado o SMA e é responsável pela cognição, isto é, pela deliberação e execução de objetivos e intenções [Souza de Jesus et al. 2022]. A camada de firmware, hospedada em um ou mais microcontroladores [Lazarin and Pantoja 2015, Pantoja et al. 2016, Souza de Castro et al. 2022], é responsável por receber as mensagens do SMA para atuar no ambiente e envia as informações do ambiente para o SMA [Souza de Jesus et al. 2022]. Dessa forma, em um SMA embarcado, existem duas camadas de processamento: aquela responsável pela cognição dos agentes (onde o SMA está localizado) e aquela responsável pela camada de sensoriamento e atuação

(firmware).

É cabível ressaltar, ainda, que a construção de um SMA embarcado requer conhecimento em diferentes campos da programação e conceitos de eletrônica. Além disso, o desenvolvimento de software para equipamentos embarcados com SMA, mesmo seguindo métodos e especificações, nem sempre é uma tarefa fácil. Muitas vezes, depende-se de diferentes ferramentas para construir, compilar e analisar o código escrito em busca de erros de implementação, falhas de sincronia, problemas de alocação de memória, erros de cálculo, entre outros problemas possíveis [Schimuneck 2014]. Logo, para reduzir essas dificuldades que os designers de SMA atualmente enfrentam, principalmente para integrar e gerenciar diferentes IDEs na tentativa de incorporar e acompanhar o sistema, pode-se concluir que a utilização de uma IDE especializada é uma solução apropriada.

Portanto, tendo em consideração todo progresso que as IDEs trouxeram para a programação de software, analogamente, a chonIDE visa atender às necessidades das camadas de raciocínio e de firmware do SMA Embarcado, através da centralização das IDEs voltadas para essas camadas em um único ambiente de desenvolvimento, facilitando para os projetistas, desse modo, a embarcação do SMA. Considerando as demandas dos designers de SMA Embarcados, a chonIDE foi criada para ser um ambiente que reúne as ferramentas intrínsecas à construção de um SMA Embarcado em uma única IDE com todas as configurações indispensáveis já executadas, permitindo, dessa maneira, que o projetista não se distraia com a preparação do ambiente e concentre seu tempo e trabalho no projeto em si. Outrossim, a embarcação e o monitoramento do SMA, com a implementação da chonIDE, podem ser efetuados remotamente, dispensando, assim, ligações por fios que tornavam esse processo limitado e inviável dependendo da aplicação.

Independentemente da função executada por um SMA Embarcado, sua estrutura é dividida em dois conjuntos de componentes fracamente acoplados: um conjunto de componentes de hardware que inclui uma unidade central de processamento, normalmente na forma de um microcontrolador; e uma série de programas de software, normalmente incluídos como firmware que conferem funcionalidade ao hardware. Na tela principal da chonIDE (Figura 1) é possível gerenciar e desenvolver a cognição do SMA no qual os agentes se relacionam, e a lógica do firmware, que transforma os sinais analógicos ou digitais em informação [Souza de Jesus et al. 2022] para comandar os sensores e atuadores através do microcontrolador, possibilitando, assim, a interação com o ambiente no qual o dispositivo está inserido, conforme as deliberações dos agentes.

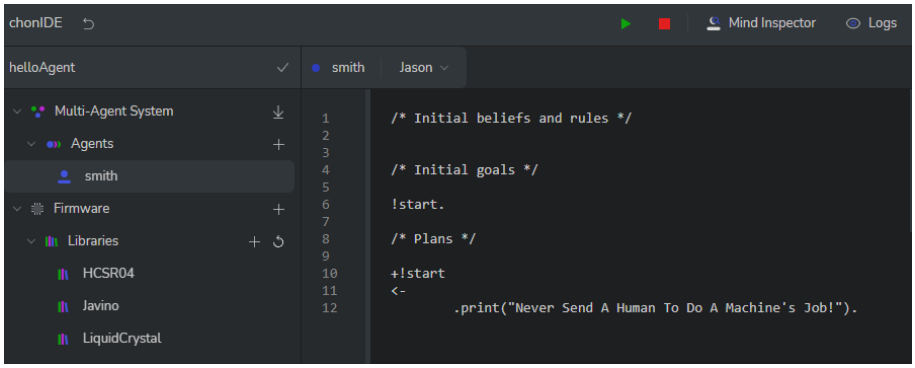

Figura 1. A tela principal da chonIDE.

A chonIDE, em sua versão atual, oferece a maneira tradicional de operação de uma IDE, na qual o usuário pode instalá-la em seu computador pessoal para desenvolver, enviar, iniciar ou interromper a execução do raciocínio, ao transmiti-lo para rodar em um protótipo robótico, e desenvolver, compilar ou implantar o firmware. Uma forma alternativa de operá-la, é acessando-a diretamente do protótipo, quando ela está rodando embarcada, via celular ou navegador.

3. Trabalhos Relacionados

Um trabalho sobre Ambientes de Desenvolvimento Integrado no apoio ao ensino da linguagem de programação Haskell foi considerado para justificar os critérios de análise que serão utilizados neste trabalho. Neste caso, a IDE Eclipse foi a mais recomendada para iniciantes, segundo diversas características julgadas desejáveis do ponto de vista de um usuário que já tem experiência em programação e está iniciando na linguagem Haskell. As particularidades citadas são: instalação, plataformas, licença de uso, manual do usuário, comunidade de usuários, comunidade de desenvolvedores, interface, compiladores ou interpretadores, depurador, editor, checagem de sintaxe, realce de sintaxe, autocompletar, ajuda rápida e indentador [Russi and Charão 2011].

Apesar de o Eclipse não servir para a mesma finalidade que a chonIDE, é aceitável utilizá-la na qualidade de modelo, pois o foco está na estrutura genérica de uma IDE e nas facilidades que essa mesma estrutura pode fornecer durante a programação, e não no propósito do desenvolvimento propriamente dito. Logo, ainda que o Eclipse não seja um ambiente para programação de SMA Embarcados, como é a chonIDE, é válido utilizá-lo como referência. Nessa mesma perspectiva, o VSCode e o IntelliJ foram considerados adequados, já que são ambientes voltados para a linguagem Java, assim como o Eclipse, e também são muito explorados no mercado de trabalho.

A partir da Tabela 1, este trabalho torna evidente o quanto a chonIDE ainda pode e precisa evoluir em versões futuras. Entretanto, mesmo em suas versões iniciais, já facilita a programação de agentes enquanto ocupa papel pioneiro no desenvolvimento de SMA Embarcados em um ambiente integrado.

Seguindo essa lógica, antes da análise da tabela, é importante expôr uma explicação dos critérios de análise envolvidos, retirados do artigo mencionado no início desta seção, para enriquecer a compreensão do leitor. A instalação do IDE deve ser fácil e rápida, para o usuário poder começar a utilizá-la o mais breve possível. É interessante que o IDE funcione nas plataformas de hardware/software mais comuns, incluindo Windows, Linux e MacOS. IDEs distribuídas sob licenças de software livre são preferíveis, por serem mais acessíveis. É desejável que o manual seja o mais completo possível, para o usuário poder sanar as dúvidas iniciais sem ter que pesquisar em fóruns. Uma comunidade numerosa é um ponto positivo, por facilitar a resolução de problemas que possam surgir. Geralmente, IDEs desenvolvidas por grupos trazem atualizações mais rapidamente que os desenvolvidos por uma só pessoa. A interface deve ser intuitiva, limpa e amigável.

É cobiçado que o depurador esteja integrado para facilitar a localização de problemas de lógica. Um editor poderoso com várias funcionalidades pode ajudar muito, por exemplo, com numeração de linhas para localização de erros reportados, múltiplas abas para acelerar a visualização de vários arquivos, linha atual destacada para facilitar a localização no arquivo, dentre outros recursos considerados abaixo. Checagem de sin-

taxe é um recurso muito útil, por avisar quando há algo errado no código antes que o usuário acione o compilador ou interpretador. Realce de sintaxe é um recurso importante, por mostrar o código de forma que fica fácil de visualizar e localizar funções, variáveis, constantes, strings, operadores, entre outros. Autocompletar auxilia na digitação do código, diminuindo os erros cometidos. Ajuda rápida são dicas mostradas quando o mouse aponta para determinadas regiões do código. Indentador é responsável pela formatação do código, auxiliando na sua organização e facilitando sua leitura [Russi and Charão 2011].

	chonIDE	Eclipse	VSCode	IntelliJ
Facilidade de Instalação	●●●○○	●●●●●	●●●●●	●●●●○
Multiplataforma	✗	✓	✓	✓
Multilinguagem	✗	✓	✓	✓
Livre para Uso	✓	✓	✓	✓
Interface clean	✓	✓	✓	✓
Debugger	✗	✓	✓	✓
Checagem de Sintaxe	✗	✓	✓	✓
Realce de Sintaxe	✗	✓	✓	✓
Autocompletar	✗	✓	✓	✓
Ajuda Rápida	✗	✓	✓	✓
Múltiplas Abas	✗	✓	✓	✓
Manual de Usuário	✗	✓	✓	✓
Visualizador de Arquivos	✓	✓	✓	✓
Contador de Linhas	✓	✓	✓	✓
Linha Atual Destacada	✗	✓	✓	✓
Indentação	✗	✓	✓	✓

Tabela 1. Comparativo de características entre as IDEs.

4. O Novo Protótipo da chonIDE

A partir da análise e comparação realizada entre a chonIDE e outras IDEs mais populares, foi possível mapear funcionalidades-base que são importantes para o desenvolvedor. O novo protótipo da chonIDE visa integrar essas principais necessidades para facilitar o processo de desenvolver e pensar um SMA.

Foram observadas três características que são comuns e consideradas como mais importantes inicialmente, seriam um console para visualização das informações de saída textuais, um depurador para visualização e acompanhamento das informações em tempo de execução e o versionamento através do Git. Essas soluções já existem de forma não integrada à IDE, sendo as duas primeiras providas pelo nativamente pelo Jason.

A disponibilização dessas funcionalidades na dinâmica de IDE possibilitaria que o desenvolvedor não precisasse sair da tela de desenvolvimento do SMA e tivesse a disposição esses recursos de visualização de informações de versionamento de forma limpa, intuitiva e prática, como proposto na Figura 2.

• ***Console integrado: Agent Tracer:*** O interpretador do Jason [Bordini et al. 2007] oferece uma interface interativa para visualização do SMA em execução, incluindo uma tela de registro de eventos e mensagens (logs) e outras ferramentas. Um dos recursos do chonOS — o sistema operacional para desenvolvimento SMA embarcado que suporta a IDE

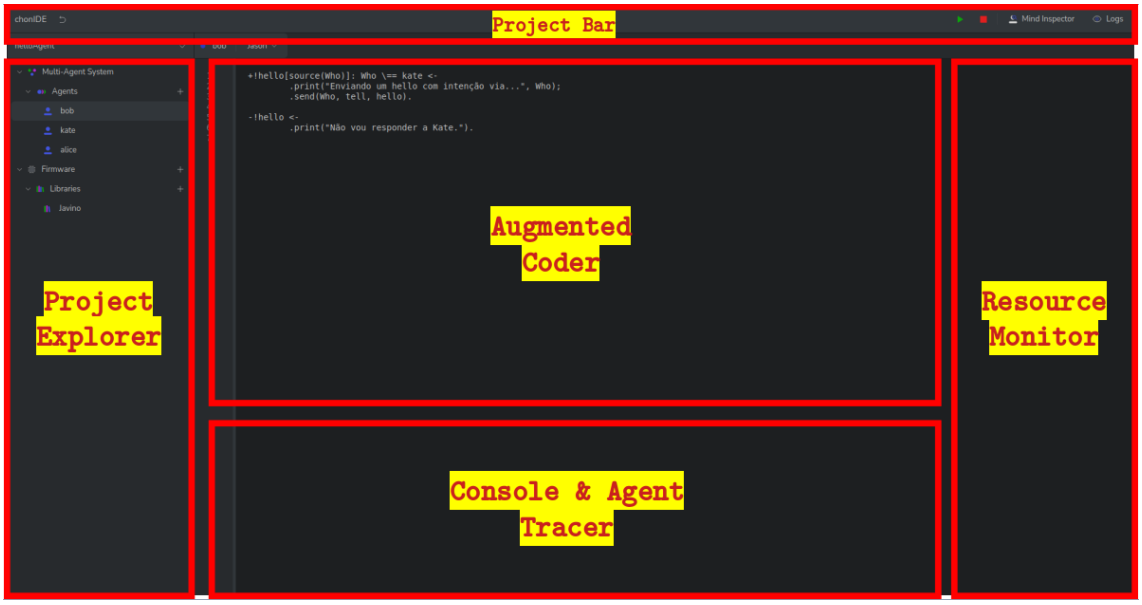

Figura 2. Os novos componentes da chonIDE.

— si captura o conteúdo textual dos logs e o escreve temporariamente em disco. Posteriormente, esse conteúdo é lido e disponibilizado em uma página web — por meio de uma aplicação chamada Wtee, escrita em Python — para a visualização desses registros em contextos embarcados e se mostrando uma solução mais reutilizável.

A integração com a IDE foi simples, a partir do aproveitamento desse recurso e visto que ambas as soluções já estão no contexto web. Ao incorporar a página do Wtee na interface da IDE, foi possível criar o componente gráfico Agent Tracer, permitindo então que o usuário não precise se deslocar da interface de desenvolvimento para visualizar os logs do SMA, simplificando e agilizando o processo de desenvolvimento.

• ***Depurador integrado: Mind Inspector integrado:*** O recurso de depuração de um SMA é existente e nativo do Jason, nomeado de Mind Inspector. É uma estrutura de páginas que exibe crenças, intenções e outras informações dos agentes. Inicialmente, na página principal, é exibida uma lista dos agentes em execução, e quando o usuário seleciona um agente, é redirecionado para uma página que se atualiza constantemente com novas informações dos ciclos de raciocínio. No entanto, essa estrutura não é integrada a IDE. Não era prático para o usuário alternar entre abas do navegador para conferir o estado do agente e funcionalmente também não era viável encaixar essa estrutura na interface do sistema. Para encontrar uma solução para essa integração, foi necessário entender como o Mind Inspector realizava esse fluxo e dispunha as informações. Para isso, foi realizado um estudo da arquitetura execução e controle do SMA no código-fonte do Jason e de seus conceitos principais, como eram dispostas crenças, intenções e planos no agente, como tudo se relacionava e onde se intercediam, além do funcionamento durante cada ciclo de execução. Como resultados do estudo, foram criados algoritmos de extração dessas informações em tempo de execução e conversão em modelos de dados diretos e intuitivos. Uma vez que os modelos de dados puderam ser instanciados com informações, a próxima etapa foi disponibilizá-los. Foi desenvolvido um mecanismo de histórico aplicável a todo agente, que armazena o estado do agente — o modelo de dados preenchido — em deter-

minado ciclo de raciocínio, e uma API Web consumida para retornar o estado do ciclo mais recente de todos os agentes do sistema ou para um agente e ciclo específico. Essa API é consumida pelo novo componente gráfico Mind Inspector na interface da IDE, localizado na barra lateral direita, que exibe, inicialmente, o estado do ciclo mais recente de todos os agentes. O usuário também pode alternar entre os ciclos de um agente específico, consolidando assim a proposta de mecânica de depuração orientada pelo ciclo do agente.

• ***Versionamento de Código:*** Devido à sua natureza distribuída e à necessidade de gerenciar múltiplas versões do projetos, os SMA Embarcados requerem um controle preciso do código-fonte. Nesse contexto, o versionamento de código desempenha um papel essencial, facilitando a colaboração entre os membros da equipe, o rastreamento de alterações e a preservação da integridade do código ao longo do desenvolvimento. A implementação proposta para a chonIDE do sistema de versionamento de código Git facilitará o processo de desenvolvimento interno e a integração com outras IDEs que suportam o desenvolvimento de SMA Embarcados. O Git é amplamente utilizado e suportado por várias ferramentas de desenvolvimento, permitindo uma colaboração eficiente entre desenvolvedores que utilizam diferentes ambientes de desenvolvimento. Essa interoperabilidade é fundamental para melhorar a padronização e a eficiência no desenvolvimento de sistemas complexos.

Para viabilizar o versionamento de código, propomos uma reestruturação das pastas e diretórios dos projetos dentro da IDE. Essa reorganização será fundamental para garantir que os arquivos estejam configurados corretamente para o controle de versão, facilitando o rastreamento das alterações ao longo do tempo e permitindo uma melhor integração com outras ferramentas e ambientes de desenvolvimento. Os componentes integrados à IDE permitirão aos desenvolvedores acessar e utilizar as funcionalidades do Git diretamente no ambiente de desenvolvimento. Isso incluirá a capacidade de iniciar e gerenciar repositórios Git, criar e gerenciar branches, realizar commits e sincronizar com repositórios remotos, tudo de maneira intuitiva e integrada. Em resumo, esta proposta de implementação do Git na chonIDE tem como objetivo melhorar o fluxo de trabalho de desenvolvimento, incentivando a colaboração entre diferentes ambientes de desenvolvimento e garantindo a segurança e a qualidade no desenvolvimento dos sistemas.

4.1. Componentização

No desenvolvimento de software, são pilares fundamentais a escalabilidade, o desacoplamento e a integração, os quais desempenham um papel importante em todo o processo de desenvolvimento. Em aplicações como uma IDE é importante que esses pilares sejam preservados, especialmente dada a amplitude de funcionalidades que esse tipo de ferramenta precisa oferecer, integrar em um mesmo contexto de forma coesa e garantir sua manutenção. Evidentemente, também reflete diretamente no desenvolvimento da interface e na experiência do usuário, dito isso, nesse contexto, foi adotado o desenvolvimento orientado a componentes.

Os frameworks de desenvolvimento front-end como o VUE.js, que é utilizado no desenvolvimento da IDE, um componente é uma unidade encapsulada, independente e replicável do código-fonte que representa visual ou interativamente um comportamento na interface. Deve ser aplicável em diferentes contextos ou conter um conjunto de funcionalidades e estados relacionados, configurando assim uma responsabilidade específica.

O processo de componentização foi conduzido em duas frentes principais: transformar as macrofuncionalidades da interface de desenvolvimento em componentes, a fim de facilitar a manutenção e a escalabilidade, e preparar o contexto que os une para integrar futuras funcionalidades com maior facilidade. Dessa forma, seria criado um sistema visual desacoplado, no qual as partes podem se expandir e mudar sem interferir no exterior, e novas partes podem ser integradas organicamente, permitindo também o desenvolvimento paralelo direcionados a esses macrocomponentes. O resultado desse processo foi a subdivisão do código-fonte em seis grandes componentes, cada um com seus próprios componentes internos: Barra Lateral Esquerda (Explorador), Agent Tracer, Codificador, Barra Lateral Direita (Mind Inspector e Firmware Boards) e Barra Superior. A Figura 3 apresenta a tela do novo protótipo[1][2] já com os componentes separados e implementados.

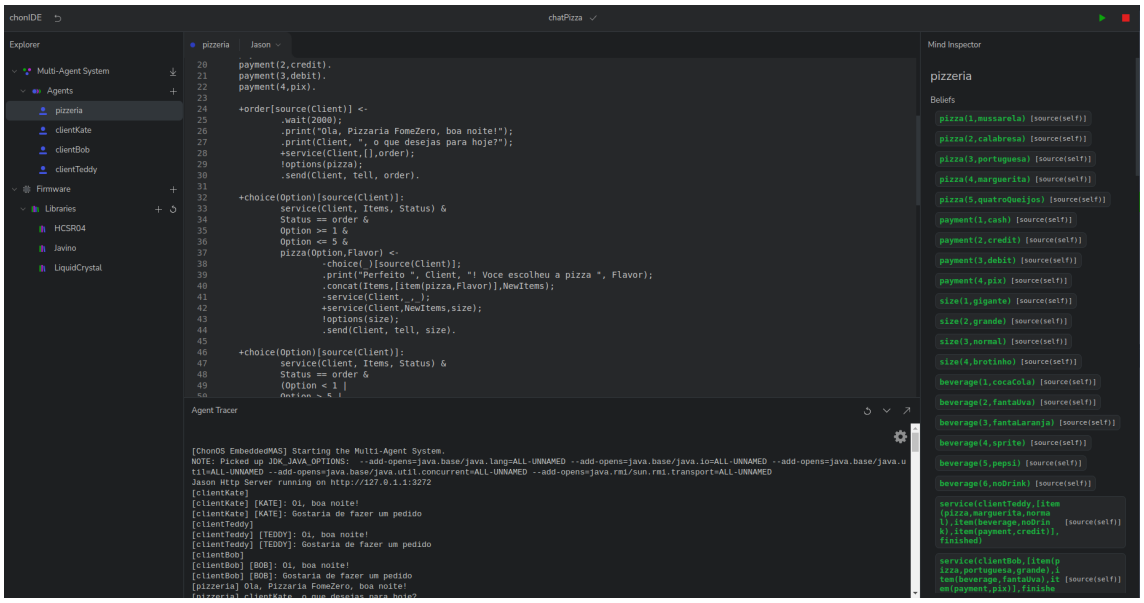

Figura 3. O protótipo em desenvolvimento da nova chonIDE.

Na Barra Lateral Esquerda, é encontrado o explorador de arquivos e pastas do projeto, onde é possível interagir com os estados relacionados ao projeto, adicionando, excluindo e modificando arquivos de agentes, firmwares e bibliotecas dentro de suas respectivas representações de pastas e arquivos. O Agent Tracer é onde o usuário pode visualizar todas as mensagens do SMA em execução no momento. O Codificador é onde ocorre a escrita e edição do código-fonte do arquivo selecionado no explorador de arquivos. Se houver um comportamento específico para o arquivo selecionado, ele será exibido no controlador de abas do codificador. Por exemplo, no arquivo do agente, é possível alternar o tipo do agente, enquanto no arquivo do firmware, é possível compilar e realizar o deploy para a placa selecionada no Firmware Boards.

A Barra Lateral Direita é alternada conforme o arquivo atualmente selecionado. Ao se tratar de um arquivo de firmware, é exibido o Firmware Boards, onde são exibidas todas as placas de firmware do sistema, e é possível selecionar para qual é desejado realizar os procedimentos de compilação ou deploy. Já ao se tratar de um arquivo de agente, é

[1]A versão online da chonIDE está disponível em https://ide.bot.chon.group:3270/chonide/login. É possível acessar utilizando o usuário *chon* e senha *ide*.

[2]O repositório da chonIDE está disponível em https://github.com/chon-group/chonIDE.

exibido o Mind Inspector, onde são exibidas todas as informações em tempo de execução de cada agente em cada ciclo, incluindo crenças, intenções, planos e outras informações. Por fim, na Barra Superior estão as informações gerais do projeto, como o nome e o indicador de salvamento automático. Além disso, são apresentados os controles gerais, como iniciar e parar o SMA do projeto e outras operações de manipulação do projeto.

5. Considerações Finais

Restrições técnicas são um desafio para os projetistas de SMA Embarcados. Um ferramental dedicado ao desenvolvimento de sistemas embarcados e distribuídos pode facilitar o processo de desenvolvimento do projetista. Dessa forma, este trabalho apresentou uma análise, restruturação e novas funcionalidades para que a chonIDE possa atenuar essas restrições em termos de agilidade e automatização de processos no desenvolvimento de SMA. Além disso, a chonIDE já traz benefícios relevantes para a integração hardware-software, através da viabilidade de projetos com a embarcação remota.

Como trabalhos futuros, pretende-se desenvolver um survey de usabilidade, aplicando a metodologia TAM (Modelo de Aceitação de Tecnologia), com uma turma da disciplina de Sistemas Especialistas do sétimo período do curso de Sistema de Informação do Cefet-RJ, a fim de sugerir pontos de aprimoramento para próximas versões da chonIDE, os quais serão refletidos em um protótipo contendo as implementações propostas, com destino a futuras avaliações, facilitando, dessa forma, ainda mais o desenvolvimento de embarcados e agentes.

Referências

Ball, S. (2002). *Embedded microprocessor systems: real world design*. Elsevier.

Bordini, R., Hübner, J., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. Wiley.

Deitel, P. J. and Deitel, H. M. (2009). *Java for programmers*. Pearson education.

Heath, S. (2002). *Embedded systems design*. Elsevier.

Hou, D. and Wang, Y. (2009). An empirical analysis of the evolution of user-visible features in an integrated development environment. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, pages 122–135.

IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Lazarin, N. M. and Pantoja, C. E. (2015). A robotic-agent platform for embedding software agents using Raspberry Pi and Arduino Boards. In *Proceedings of the 9th Workshop-School on Agents, Environments, and Applications (WESAAC 2015)*, pages 13–20, Niteroi. UFF. `http://www2.ic.uff.br/~wesaac2015/Proceedings-WESAAC-2015.pdf`.

Manoel, F. C. P. B., dos Santos Amorim, M. C. M., and Pantoja, C. E. (2022). Um metamodelo para sistemas multiagentes embarcados considerando as dimensões multiagente, estrutural e comportamental. In *Anais do XVI Workshop-Escola de Sistemas*

de Agentes, Seus Ambientes e Aplicações (WESAAC 2022), pages 61–72, Blumenau. UFSC.

Marwedel, P. (2021). *Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things*. Springer Nature.

Ojo, M. O., Giordano, S., Procissi, G., and Seitanidis, I. N. (2018). A review of low-end, middle-end, and high-end iot devices. *IEEE Access*, 6:70528–70554.

Pantoja, C. E., Stabile, M. F., Lazarin, N. M., and Sichman, J. S. (2016). ARGO: An Extended Jason Architecture that Facilitates Embedded Robotic Agents Programming. In Baldoni, M., Müller, J. P., Nunes, I., and Zalila-Wenkstern, R., editors, *Engineering Multi-Agent Systems*, pages 136–155, Cham. Springer International Publishing. https://doi.org/10.1007/978-3-319-50983-9_8.

Russi, D. F. and Charão, A. S. (2011). Ambientes de desenvolvimento integrado no apoio ao ensino da linguagem de programação haskell. *Revista Novas Tecnologias na Educação*, 9(2).

Schimuneck, T. (2014). Ambiente de desenvolvimento integrado para ensino e programação de microcontroladores da família mcs51.

Sichman, J. S., Demazeau, Y., and Boissier, O. (1992). When can knowledge-based systems be called agents? In *Anais do SBIA 92*, pages 172–185, Rio de Janeiro. SBC.

Souza de Castro, L. F., Manoel, F. C. P. B., Souza de Jesus, V., Pantoja, C. E., Pinz Borges, A., and Vaz Alves, G. (2022). Integrating Embedded Multiagent Systems with Urban Simulation Tools and IoT Applications. *Revista de Informática Teórica e Aplicada*, 29(1):81–90. DOI: https://doi.org/10.22456/2175-2745.110837.

Souza de Jesus, V., Lazarin, N. M., Pantoja, C. E., Manoel, F. C. P. B., Alves, G. V., Ramos, G., and Filho, J. V. (2022). Proposta de uma ide para desenvolvimento de sma embarcados. In *Anais do XVI Workshop-Escola de Sistemas de Agentes, Seus Ambientes e Aplicações (WESAAC 2022)*, pages 49–60, Blumenau. UFSC.

Souza de Jesus, V., Mori Lazarin, N., Pantoja, C. E., Vaz Alves, G., Ramos Alves de Lima, G., and Viterbo, J. (2023). An IDE to Support the Development of Embedded Multi-Agent Systems. In Mathieu, P., Dignum, F., Novais, P., and De la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection*, pages 346–358, Cham. Springer Nature Switzerland. DOI: https://doi.org./10.1007/978-3-031-37616-0_29.

Wooldridge, M. (2000). Intelligent Agents. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1st edition.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John wiley & sons.