Revista Eletrônica de Iniciação Científica em Computação, 2025, 23:1
ISSN: 3085-8461 • doi: 10.5753/reic.2025.5320

Artigo de Pesquisa/Research Paper

Avaliação de Melhorias Funcionais para um IDE de SMA Embarcados

Evaluation of Functional Improvements for an Embedded MAS IDE

Elaine M. P. Siqueira [Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET-RJ) | *elaine.siqueira@aluno.cefet-rj.br*]
Gabriel Ramos A. Lima [Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET-RJ) |*gabriel.ramos@aluno.cefet-rj.br*]
Thácito Raboni [Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET-RJ) |*thacito.medeiros@aluno.cefet-rj.br*]
Carlos Eduardo Pantoja [Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET-RJ) |*carlos.pantoja@cefet-rj.br*]
Nilson Mori Lazarin [Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET-RJ) |*nilson.lazarin@cefet-rj.br*]

Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET-RJ), R. Miguel Ângelo, 96, Maria da Graça, Rio de Janeiro, RJ, 20785-220, Brasil.

Resumo. O desenvolvimento de Sistemas Multiagentes (SMA) Embarcados sofre com restrições que tornam esse processo pouco intuitivo. O uso de um ferramental especializado pode facilitar o trabalho do projetista. Este artigo apresenta uma análise, restruturação e novas funcionalidades para que um ambiente dedicado a SMA embarcado possa permitir agilidade e automatização de processos na engenharia de software de SMA. Foi realizado uma pesquisa de usabilidade, aplicando o Modelo de Aceitação de Tecnologia, a fim de localizar melhorias refletidas em um novo protótipo contendo as implementações propostas.

Abstract. The development of Embedded Multi-Agent Systems (MAS) suffers from constraints that make this process unintuitive. The use of specialized tools can facilitate the developer's work. This article presents an analysis, restructuring, and new functionalities so that an environment dedicated to embedded MAS can allow agility and automation of processes in MAS software engineering. A usability research was conducted, applying the Technology Acceptance Model, to identify improvements reflected in a new prototype containing the proposed implementations.

Palavras-chave: Agentes, Sistemas Multiagentes, Sistemas Multiagentes Embarcados, IDE

Keywords: Agents, Multi-Agent Systems, Embedded Multi-Agent Systems, IDE

Recebido/Received: 12 December 2024 • **Aceito/Accepted:** 02 February 2025 • **Publicado/Published:** 24 February 2025

1 Introdução

Os programadores passam grande parte do seu tempo interagindo com Ambientes de Desenvolvimento Integrado (IDEs), que ajudam a aumentar sua produtividade através da automatização de parte do seu trabalho. Como qualquer software útil, IDEs vem se tornando mais poderosos e utilizáveis à medida que funcionalidades são adicionadas e questões de usabilidade são abordadas [Hou and Wang, 2009].

Apesar de uma variedade de IDEs já ter sido absorvida pelo mercado tecnológico, como o Eclipse, IntelliJ e VS-Code, esses ferramentais populares ainda não suportam a embarcação de Sistemas Multiagentes (SMA), devido a sua natureza distribuída, a qual necessita equipagem especifica. Um SMA é um sistema composto por agentes cognitivos da Inteligência Artificial (IA) os quais são dotados de autonomia, proatividade, comunicabilidade e sociabilidade [Wooldridge, 2009]. Esses agentes são situados em ambientes explorados tanto de forma endógena (representações virtuais que agentes podem manipular) quanto exógena (o mundo real em si) [Ricci *et al*., 2012]. Um SMA Embarcado considera um conjunto de agentes cooperando para gerenciar recursos (sensores e atuadores) em um dispositivo físico capaz de interagir diretamente com outros no mundo real [Brandão *et al*., 2021; Pantoja *et al*., 2023].

Um IDE bem projetado deve auxiliar os programadores a automatizar tarefas rotineiras e reduzir a carga cognitiva desses designers, para que eles possam se concentrar nos aspectos criativos do processo de engenharia de software [Hou and Wang, 2009]. Nesse sentido, o chonIDE [Souza de Jesus *et al*., 2023], apesar de ter a capacidade de assistir desde a concepção lógica dos SMA até seu monitoramento após a incorporação, não possui funcionalidades suficientes voltadas a tornar a experiência do usuário mais eficaz. Por essa lógica, este trabalho realiza um comparativo entre o ChonIDE e alguns IDEs de mercado, a fim de ilustrar as funcionalidades que ele possui, específicas para o contexto de SMA embarcado, das quais essas outras IDEs carecem, e também de identificar possíveis implementações para evoluí-lo em termos de experiência de usuário, com base nessas IDEs já consolidadas.

Outrossim, ainda que na literatura já exista uma arquitetura sólida sobre esse tipo de sistema [Pantoja *et al*., 2016], deve-se considerar que a construção de um SMA embarcado requer conhecimento em diferentes campos da programação e conceitos de eletrônica; e se encontra em estágios embrionários de desenvolvimento e estabelecimento de etapas de engenharia de software. Nessa perspectiva, tanto o ChonIDE quanto os demais já citados, não contemplam parte do processo de engenharia como *deploy*, testes automatizados, diagnósticos e comunicabilidade, além disso, as atuais funcionalidades, como complementação de código e destacamento de sintaxe, não estão suficientemente avançadas para linguagens voltadas a agentes. Portanto, uma pesquisa de usabilidade do ChonIDE foi aplicada em turmas regulares de cursos

de graduação e pós-graduação de instituições brasileiras e em mini-cursos apresentados durante o último ano, para que, através da análise dos resultados, seja possível direcionar os próximos pontos de aperfeiçoamento e avanços funcionais do ChonIDE.

O objetivo deste artigo é apresentar uma discussão estendida[1] dos recursos e aprimoramentos possíveis do ferramental para SMA Embarcados [Siqueira *et al*., 2024], por meio da investigação das principais funções dos IDEs atuais em contraste com as que são disponibilizadas pelo ChonIDE e do estudo de usabilidade, a fim de sugerir a introdução de novas funcionalidades e melhorias nele e, consequentemente, enriquecer a experiência do desenvolvedor e aumentar a eficácia no processo de engenharia de software de SMA Embarcados. Ademais, também é contribuição deste trabalho um comparativo entre as versões do protótipo do ChonIDE pré e pós as implementações de alguns dos recursos fundamentados nas análises citadas, com a finalidade de examinar a relevância das alterações sugeridas.

Este trabalho está estruturado da seguinte forma: a Seção 2 discorre sobre conceitos que compõem a compreensão do campo de SMA Embarcado; a Seção 3 apresenta uma fundamentação para a tabela comparativa das limitações e diferenciais do chonIDE em relação a outros três IDEs já mencionados; a Seção 4 detalha a pesquisa de usabilidade realizada; a Seção 5 exibe o novo protótipo do software abordado neste documento; a seção 6 disserta sobre um teste entre as versões do chonIDE antes e depois da implementação de algumas mudanças levantadas neste escrito e, finalmente, a Seção 7 faz as considerações finais e indica trabalhos posteriores.

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].

Devido à habilidade social, que caracteriza os agentes, isto é, capacidade de realizar interações com outros agentes [Alvares and Sichman, 1997], 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, 2002], isto é, realizar tarefas complexas [Wooldridge, 2009].

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. 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 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 [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 [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 um IDE especializada é uma solução apropriada.

Portanto, tendo em consideração todo progresso que os IDEs trouxeram para a programação de software, analogamente, o chonIDE visa atender às necessidades das camadas de raciocínio e de firmware do SMA Embarcado, através da centralização dos IDEs voltados 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, o chonIDE foi criado para ser um ambiente que reúne as ferramentas intrínsecas à construção de um SMA Embarcado em um único 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

[1] Este trabalho é uma extensão daquele apresentado no WESAAC 2024, submetido a REIC, a convite, como melhores trabalhos em graduação do evento.

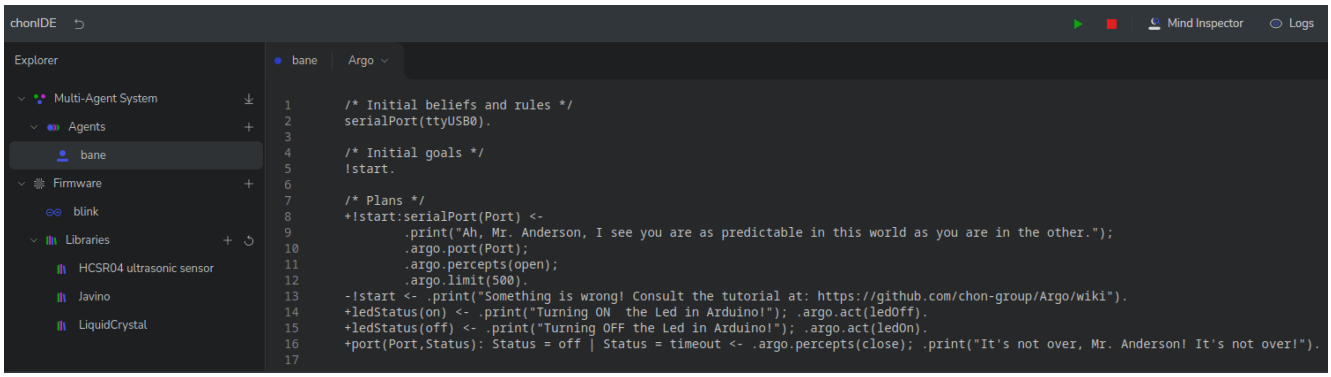

Figura 1. A tela principal do chonIDE.

e trabalho no projeto em si. Outrossim, a embarcação e o monitoramento do SMA, com a implementação do 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 do chonIDE Figura 1 é possível gerenciar (na barra lateral esquerda) e desenvolver no editor de texto a cognição do SMA no qual os agentes se relacionam e a lógica do firmware, que transforma os sinais analógicos ou digitais em informação [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.

O chonIDE, em sua versão atual, oferece a maneira tradicional de operação de um IDE, no qual o usuário pode instalá-lo em seu computador pessoal para desenvolver, enviar, iniciar ou interromper (na barra superior) 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-lo, é acessando-o diretamente do protótipo, quando ele está rodando embarcado, via celular ou navegador.

3 Comparativo entre o ChonIDE e IDEs de Mercado

Um trabalho sobre IDEs no apoio ao ensino da linguagem de programação Haskell foi considerado para justificar os critérios de análise que serão utilizados neste trabalho. Neste caso, o IDE Eclipse foi a mais recomendada para iniciantes, segundo diversas características 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 o chonIDE, é aceitável utilizá-lo de referência, pois o foco está na estrutura genérica de um 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 o chonIDE, é válido que ele sirva de modelo. 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 o 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, deve-se considerar que alguns critérios de análise são importantes em IDEs [Russi and Charão, 2011]: 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 distribuídos sob licenças de software livre são preferíveis, por serem mais acessíveis. É desejável que o manual seja o mais completo possível, para o usuário po-

Tabela 1. Comparativo de características entre as IDEs.

	chonIDE	Eclipse	VSCode	IntelliJ
Facilidade de Instalação	●●●○○	●●●●●●●●●●●●●●		
Multiplataforma	✗	✓	✓	✓
Multilinguagem	✗	✓	✓	✓
Livre para Uso	✓	✓	✓	✓
Interface limpa	✓	✓	✓	✓
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	✗	✓	✓	✓

der 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 desenvolvidos 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 com várias funcionalidades pode ajudar 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. A checagem de sintaxe é um recurso útil em 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 sua organização e facilitando a leitura [Russi and Charão, 2011].

4 Pesquisa de Usabilidade

As descobertas sobre como a usabilidade evolui pode ser de particular interesse para os desenvolvedores de IDE que buscam soluções e técnicas para melhorar seu produto [Hou and Wang, 2009]. Nessa perspectiva, uma pesquisa com 25 participantes foi aplicada em turmas das disciplinas de graduação de Sistemas Especialistas do CEFET-RJ (campus Maria da Graça) e IA Distribuída e Embarcada (campus Nova Friburgo), do curso de mestrado e doutorado em Computação da UFF na disciplina de Introdução a Sistemas Multiagentes e de minicursos de SMA Embarcados ocorridos em workshops e conferências, com o intuito de compreender a visão do usuário a respeito do chonIDE.

Conduzida sob a metodologia do Modelo de Aceitação de Tecnologia (TAM), que avalia a aceitação de uma nova tecnologia com base na Percepção de Facilidade de Uso (PEOU) e na Percepção de Utilidade (PU) que o usuário tem da ferramenta, essa pesquisa de usabilidade busca levantar pontos de melhoria para serem aplicados em um novo protótipo do software por meio de um questionário online.

As perguntas realizadas no teste estão voltadas a entender dificuldades que o usuário enxerga no fluxo de uso do IDE, além de identificar barreiras da interface, como falta de fluidez ou demora para realizar uma tarefa, a fim de garantir a consistência do produto.

Entre as questões levantadas estão:

- *O quanto você concorda que as funcionalidades do chonIDE mencionadas a seguir facilitam o desenvolvimento de SMA Embarcado?*
- *Você conseguiria identificar alguma outra funcionalidade além das já mencionadas? Se sim, qual(is) seria(m)?*
- *O quanto você concorda que a usabilidade do chonIDE:* (afirmações)
- *Você conseguiria atribuir alguma outra característica da usabilidade além das já mencionadas? Se sim, qual(is) seria(m)?*
- *Em sua opinião, quais são os pontos fortes da ferramenta apresentada (chonIDE)?*
- *Em sua opinião, quais são os pontos fracos da ferramenta apresentada (chonIDE)?*
- *Com base em suas respostas anteriores e, em sua opinião, você conseguiria propôr melhorias para as questões apresentadas? Se sim, quais?*

O perfil dos investigados inclui 20 alunos de graduação e 5 professores ou alunos de pós-graduação, dos quais 23 dizem ter familiaridade moderada ou superior com programação estruturada, 19 se consideram profissionais ou avançados em programação orientada a objeto (POO) e 6 se consideram moderados ou profissionais em programação orientada a agentes (POA). Além disso, 19 tem nenhuma ou pouca familiaridade com desenvolvimento de SMA, 23 tem familiaridade moderada ou inferior com desenvolvimento de sistemas embarcados e todos tem familiaridade moderada ou inferior em desenvolvimento de SMA Embarcado.

A coleta dos dados de PU e PEOU se dá por meio de afirmativas que destacam as possíveis utilidades que o software pode atingir, nelas os participantes do formulário marcam 1 entre 5 níveis de concordância (desde discordar totalmente até concordar plenamente), o que possibilita que uma média ponderada seja gerada para cada frase de acordo com a quantidade de respostas em cada nível de concordância de uma mesma afirmativa. Além disso, campos de texto livre estão dispostos para manifestar pontos fortes, fracos e sugerir alguma utilidade que não tenha sido descrita pelo questionário, tendo esta sido ou não identificada no IDE.

Como pode ser visto na Tabela 2, a média de resultados de PEOU mostra que 73% dos pesquisados considera a interface limpa e compreensível e 68% acha que ela é fácil de aprender. A maioria dos comentários realizados por eles sobre o visual da interface endossam esses percentuais, como: "*Design moderno*", "*Fluidez, menus simples.*", "*A interface é limpa, é fácil encontrar o que estou procurando e a importação e exportação de projetos é simples.*", "*É simples e fácil de entender e manusear.*", "*Design agradável e intuitivo.*", "*Facilidade de criar projetos e agentes.*", entre outros.

Por outro lado, na Tabela 3, a média de resultados de PU mostra que os avaliados que consideram o IDE útil: no processo de desenvolvimento de SMA Embarcado são 76,8%, no aprendizado do paradigma de POA são 74,4% e no aprendizado da linguagem de POA suportada são 72%. Algumas das observações feitas por eles confirmam: "*Boa usabilidade e fácil integração com softwares e hardwares.*", "*Desenvolvimento de SMA de forma rápida.*", "*É útil para o aprendizado.*", "*Envio do código para o sistema embarcado de forma transparente e possui todas as funcionalidades integradas.*", "*Reduz o esforço cognitivo para desenvolvimento*", etc.

Os feedbacks específicos de funcionalidades já existentes no chonIDE possuem as seguintes percepções: "*Não interrompe corretamente a operação dos agentes.*", "*Dificuldade na visualização do código, dificuldade em descobrir exceções ocorridas.*", "*Falta de paralelismo para edição simultânea do código.*", "*Mind Inspector não é tão apresentável ao usuário.*", "*Ausência de console.*", e mais. Além disso,

Tabela 2. Análise da PEOU do ChonIDE.

Pontuação PEOU	Discordo Totalmente (1 pt)	Discordo Parcialmente (2 pts)	Nem concordo nem discordo (3 pts)	Concordo Parcialmente (4 pts)	Concordo Plenamente (5 pts)	Média por afirmativa (0-5 pts)
Interface clara e compreensível	0	5	4	10	6	3,68
Interface fácil de aprender	2	3	9	5	6	3,4

Tabela 3. Analise da PU do chonIDE.

Pontuação PU	Discordo Totalmente (1 pt)	Discordo Parcialmente (2 pts)	Nem concordo nem discordo (3 pts)	Concordo Parcialmente (4 pts)	Concordo Plenamente (5 pts)	Média por afirmativa (0-5 pts)
Útil no processo de desenvolvimento de SMA Embarcado	0	2	6	11	6	3,84
Útil no aprendizado do paradigma de POA	2	2	4	10	7	3,72
Útil no aprendizado da linguagem de POA suportada (Agent Speak)	3	2	4	9	7	3,6

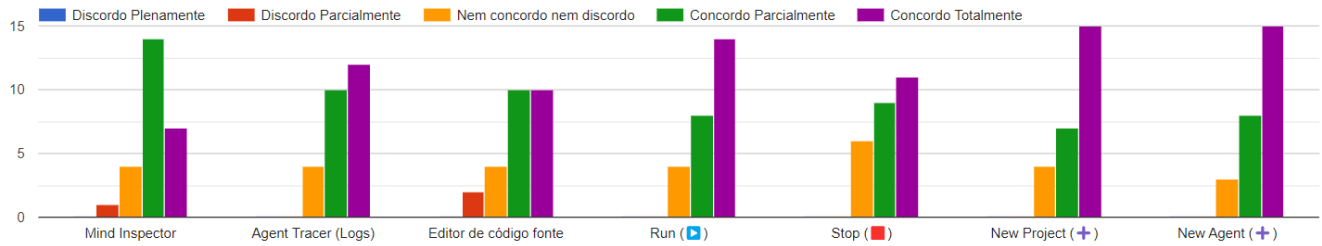

Figura 2. Analise das funcionalidades do ChonIDE no Desenvolvimento de SMA Embarcado

incômodos sobre funções desejadas pelo usuário foram expostos: "*Não tem 'autocomplete' [...].*", "*Não tem 'syntax highlighting'.*", "*Suporte limitado a plugins e extensões.*", e também sugestões de aprimoramento: "*Verificação de sensores de entrada de forma visual*", "*Um manual integrado.*", "*Integração com Git, debug e mais teclas de atalho.*", entre outros.

Portanto, de acordo com a Figura 2, torna-se notável a insatisfação mais expressiva do usuário com duas ferramentas — o Mind Inspector e o Editor de Código — já que são as únicas que tem nota 2 entre as votações, correspondente a "discordo parcialmente que a funcionalidade facilite o processo de desenvolvimento de SMA Embarcado".

Outrossim, diferentemente das demais analisadas, somente o primeiro não tem a última coluna (nota 5) do gráfico maior que as demais, já o segundo, apesar de tê-la, essa está muito próxima da de nota 4 e seu gráfico está com as barras bem equilibradas, o que faz concluir que as opiniões sobre essa funcionalidade estão ainda bastante distribuídas. Fundamenta-se, então, que o novo protótipo do IDE tratada deva dar prioridade a aperfeiçoar essas funções, com a finalidade de melhorar a experiência do usuário em sua plataforma.

5 Novo Protótipo

A partir da análise e comparação realizadas entre o chonIDE e os IDEs mais populares, é possível mapear funcionalidades que se destacam por facilitar o desenvolvimento de software, na medida em que permitem que o programador se concentre exclusivamente na construção da solução. Nesse sentido, é proposto um novo protótipo, considerando os principais questionamentos levantados pela pesquisa de usabilidade, a partir de modificações arquiteturais e conceituais da atual ferramenta.

O novo protótipo proposto visa integrar uma experiência de desenvolvimento em *single-screen*: um console de exibição de saídas de agentes; um depurador de estados mentais dos agentes integrados ao ferramental, e modificações arquiteturais para permitir a componentização de novas funcionalidades, como ilustrado na Figura 3.

Uma arquitetura *single-screen* permite construir um SMA Embarcado sem a necessidade de alternar entre janelas ao executar tarefas relativas a programação e design de tais sistemas. Atualmente, dado as diversas camadas de intervenção que um desenvolvedor precisa atuar (hardware, firmware, agentes e sistema operacional) ainda é necessário navegar entre diversos IDE, o que dificulta a integração das partes necessárias para prover um SMA com interfaceamento com dispositivos adequadamente. Depurar um agente requer alternação entre visualizações entre código e depura-

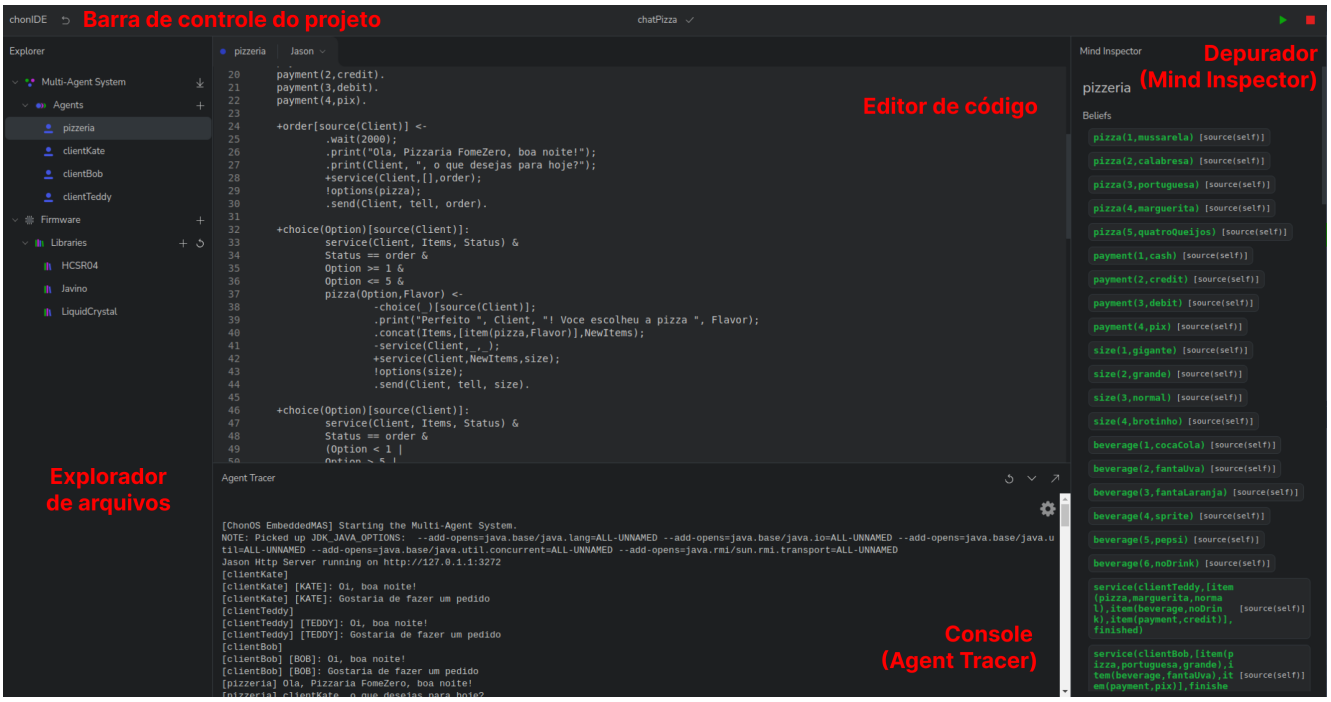

Figura 3. Novo protótipo

ção. Ademais, o chonIDE não está preparada para a componentização necessária para novas funcionalidades sem uma refatoração. Dessa forma, neste trabalho, duas funcionalidades foram expandidas e integradas ao ferramental:

- **Console (Agent Tracer)**: o console tem como objetivo principal dispor as saídas exibíveis dos agentes — registros de eventos e mensagens — no decorrer da atividade do SMA, além de outras informações do sistema. Durante a execução do projeto, um serviço é responsável por capturar a saída textual do processo e gravá-la temporariamente em disco para, posteriormente, outro serviço ler esse conteúdo e apresenta-lo em uma página web. Essa funcionalidade permite que as informações dos agentes e sistema sejam acessados até em contextos embarcados. Em versões anteriores, o Agent Tracer só era permitido em abas separadas, obrigando ao desenvolvedor não estar diante do código enquanto analisa as mensagens.
- **Depurador (Mind Inspector Integrado)**: O recurso de depuração de um SMA é provido nativamente pelo interpretador do Jason [Bordini *et al.*, 2007] e consiste em uma estrutura de páginas web que exibe crenças, intenções, planos em execução e outros detalhes do agente. Esse mecanismo é chamado de Mind Inspector Web e, embora cumpra sua função, não está integrado à IDE, exigindo que o desenvolvedor alterne entre as abas do navegador para monitorar o estado dos agentes. Isso prejudica a usabilidade de um ambiente integrado, na medida em que interrompe a experiência de desenvolvimento em single-screen. Diante disso, se fez necessário entender o fluxo de disponibilização das informações dessa ferramenta nativa e, a partir disso, desenvolver uma nova solução própria e integrável à IDE. Dessa forma, foi criado o Mind Inspector Integrado, que permite visualizar o estado mais recente de todos os agentes bem como consultar o estado de um determinado agente em um ciclo específico e alternar facilmente entre essas dinâmicas. Esse mecanismo é viabilizado através da extração das estruturas que compõe o estado do agente e conversão delas em um modelo de dados direto, o armazenamento das instâncias da modelagem em um mecanismo de histórico e a disponibilização dessas mesmas instâncias através de uma API Web, que é consumida pelo componente gráfico da funcionalidade — localizado na barra lateral direita da interface do chonIDE.

5.1 Componentização

As novas e futuras funcionalidades presentes no protótipo do chonIDE a enquadram como um software que precisa abranger múltiplos recursos em um mesmo contexto. Nesse caso, para viabilizar a incorporação de novos recursos e a extensão dos que já existem, é necessário garantir os aspectos da integrabilidade e desacoplamento entre os componentes. Esse processo de componentização implica em uma refatoração no *front-end* do chonIDE, pensando em serviços e escalabilidades futuras.

Um componente é um módulo do código que é responsável pela renderização e forma que um conteúdo é exibido (apresentacional) ou por lidar com a lógica de negócios, gerenciamento de estados e comunicação com recursos externos (contêiner). Em destaque, é nesse último que se baseia essa metodologia, resultando na reestruturação da página de desenvolvimento do projeto por meio do encapsulamento das macro-estruturas que a compõem. Assim, se isola a funcionalidade específica do meio externo que a integra, garantindo o aspecto do desacoplamento. Essa abordagem é possível através de novas tecnologias que viabilizam a criação de interfaces reativas (e.g., Vue.js). A partir desse processo, a interface está dividida em seis componentes contêineres:

- **Explorador de Arquivos**: Na barra lateral esquerda, é responsável pelo controle dos arquivos do projeto, viabilizando sua criação, exclusão e modificação. Prevê comunicação com outros componentes futuros (e.g., versionamento de códigos).
- **Agent Tracer**: Na barra inferior do chonIDE, permite a visualização e o controle do conteúdo incorporado referente as saídas do agente de um SMA.
- **Mind Inspector**: Na barra lateral direita, exibe o estado da mente dos agentes em execução e permite uma relação visual entre essas informações e o código-fonte do projeto.
- **Placas de Firmware**: Está localizado na barra lateral direita e é exibido ao selecionar um arquivo de firmware. Permite também que o desenvolvedor selecione a placa para onde deseja realizar uma compilação ou deploy do código-fonte.
- **Codificador**: No centro, exibe o código-fonte do arquivo selecionado, permite a sua codificação e, inclusive, garante seu salvamento automático e destaques visuais a partir de outras estruturas. Está preparado para integração com serviços nativos como Servidores de Linguagem (LSP), que permitem completação de código e identificação de sintaxe.
- **Barra de controle de projeto:** Na parte superior, permite o controle de execução do projeto e a exibição do seu estado de salvamento automático.

A partir disso, consequentemente, restam às páginas (ou contêineres maiores), que integram as partes citadas acima, apenas duas responsabilidades: o gerenciamento dos recursos que são compartilhados (i.e., informações comuns na memória) e a disposição e organização visual. Para esse último caso, é atribuída uma característica flexível aos componentes filhos, desse modo, permitindo que ajustem sua forma e conteúdo em função da redistribuição realizada conforme a necessidade e a intenção do desenvolvedor. Essa característica permite que o usuário, em determinados momentos, possa dar foco específico para componentes desejados sem sair da mesma tela, dessa forma, o aspecto da integrabilidade também é preservado, pois o ambiente que abriga as funcionalidades está preparado para se adaptar de forma orgânica a adições e extensões.

6 Comparativo entre o Antigo e o Novo Protótipo

A proposta do novo protótipo do chonIDE tem como objetivo melhorar a experiência de construir um SMA embarcado por meio da integração de funcionalidades destacáveis no processo de desenvolvimento observado em IDEs do mercado. Com intuito de demonstrar esses novos recursos, nessa seção é realizada uma comparação entre a versão antiga e a nova da ferramenta, com destaque para o uso do depurador e do console.

Partindo dessa premissa, utilizando do projeto *pizzeria*, exemplo que simula um *ChatBot* com três agentes (Teddy, Bob e Kate) representando clientes que realizam pedidos a um agente que é uma pizzaria (Pizzeria) que atende a esses mesmos pedidos, foi realizado uma análise dos fluxos de utilização de ambas as versões do IDE após executar o SMA considerando a forma de utilização e acesso às respectivas novas funcionalidades (Agent Tracer e Mind Inspector).

A partir disso, os índices levados em consideração na comparação do uso dos novos recursos foram: a quantidade de cliques que o usuário precisa realizar para acessa-lo, o indicativo da existência ou não dos mesmos de forma integrada e a existência de dinâmicas de facilitação no uso deles. Dito isso, o comparativo pode ser visto na Tabela 4.

É possível perceber uma maior imersão para a experiência do usuário durante o processo de desenvolvimento, viabilizada pela integração e a melhor disposição visual e estética dos recursos, indiciado principalmente pelos itens A, C e I e também pela disponibilização de novos meios de facilitação no modo de uso das respectivas ferramentas, como os itens D e K. A nova dinâmica na depuração, indicada no item E, permite uma maior interação e associação do conteúdo mostrado com o código-fonte desenvolvido, de modo que é possível mapear visualmente a origem de determinada crença ou plano (sendo do mesmo agente ou vindo de outro através de processos de comunicação) no codificador através de um destaque no trecho código-fonte. Também está apresentada uma melhor navegabilidade das funcionalidades através da diminuição da distância entre o desenvolvedor e o recurso que o mesmo deseja acessar utilizando como métricas a quantidade de cliques nos itens F, G, H e L. Isso se justifica pela não necessidade de alternar entre as abas do navegador e a exibição imediata do estado de todos os agentes do SMA direto no componente do depurador. Os pontos apresentados evidenciam o impacto positivo da nova versão do chonIDE na construção de SMA, proporcionando uma interface *single-*

Tabela 4. Comparativo de usabilidade entre o antigo e o novo protótipo do ChonIDE.

Item	Características	Versão nova	Versão antiga
A	Depurador integrado na interface	Sim	Não
B	Crenças, intenções, planos, ações executadas e valores de variáveis na depuração	Sim	Sim
C	Design moderno e limpo para a depurador	Sim	Não
D	Facilidade em alternar entre ciclos através de botões de avançar e voltar na depuração	Sim	Não
E	Destaque do código-fonte na interação através do clique no conteúdo do depurador	Sim	Não
F	Cliues para acessar o depurador	0	1
G	Cliques para visualizar a mente de um agente específico no depurador	0	2
H	Cliques para visualizar a mente de um agente específico em um ciclo específico no depurador	1	3
I	Console integrado na interface	Sim	Não
J	Visualização de todos os logs do SMA no console	Sim	Sim
K	Botão de limpar os logs do console	Sim	Não
L	Cliques para acessar o console	0	1

screen, práticas de desenvolvimento mais fluídas e um aumento direto na produtividade do usuário. A integração de recursos como o Mind Inspector e o Agent Tracer simplificam a utilização do IDE e demonstram seu alinhamento com as necessidades dos desenvolvedores modernos.

7 Conclusão

A análise de funcionalidades realizada pela pesquisa de usabilidade, bem como a comparação do ChonIDE com outras IDEs amplamente utilizadas no mercado, como Eclipse, VSCode e IntelliJ, evidenciou a necessidade da implementação de duas funcionalidades principais para aprimorar a experiência de desenvolvimento em *single-screen*: um depurador e um console integrado. Essas melhorias destacam-se por mitigar restrições relacionadas à agilidade e à automação nos processos de programação de SMA embarcados, como evidenciado pela comparação entre a versão antiga e a nova do protótipo.

Além disso, como trabalhos futuros, ainda planeja-se explorar funcionalidades que deem suporte para a concepção de software através do IDE de forma mais eficaz, conforme indicado pela pesquisa de usabilidade, como a integração com Servidores de Linguagem (LSP) para checagem e realce de sintaxe, autocompletar de código, ajuda rápida, indentador e o versionamento de código integrado. Esse último, por sua vez, será viabilizado por meio de uma nova arquitetura de back-end que especificará uma estrutura de pastas e arquivos para os projetos e melhorias no processo de implementação de novas funcionalidades. Ademais, pretende-se aperfeiçoar o Mind Inspector, escalando-o para uma ferramenta de controle integral do SMA em execução e, dessa forma, o desenvolvedor poderá adicionar e alterar recursos, como crenças, intenções, planos e outras dinâmicas através de uma interface interativa. Portanto, será possível facilitar o processo de desenvolvimento de SMA embarcados através do ferramental abordado por este artigo.

Declarações complementares

Contribuições dos autores

Elaine Siqueira é a principal contribuidora deste trabalho, responsável pela concepção das ideias, formulação dos objetivos, redatora majoritária do manuscrito e a responsável pelo levantamento dos dados e informações discutidas. Gabriel Lima é o principal desenvolvedor da ferramenta chonIDE e realizou a implementação do novo protótipo apresentado e as experimentações finais. Thacito Raboni também participou do desenvolvimento do novo protótipo. Carlos Eduardo Pantoja supervisionou a equipe, forneceu orientação geral para o trabalho e contribuiu com a revisão e edição do manuscrito. Nilson Lazarin também contribuiu com a revisão e edição do manuscrito. Todos os autores leram e aprovaram a versão final do manuscrito.

Conflitos de interesse

Os autores declaram que não têm nenhum conflito de interesses.

Disponibilidade de dados e materiais

O questionário, as respostas, o código-fonte e um vídeo demonstrativo estão disponíveis na página de reprodutibilidade do artigo[2].

[2] `https://papers.chon.group/REIC/melhoriasFuncionais/`

Referências

Alvares, L. O. and Sichman, J. S. (1997). Introdução aos Sistemas Multiagentes. In *Anais da Jornada de Atualização em Informática - XVI JAI*, volume 97, Brasília. SBC. `https://scholar.google.com.br/scholar?cluster=17904554887101077643`.

Ball, S. R., editor (2002). *Embedded microprocessor systems: real world design*. Embedded technology series. Newnes, Amsterdam Boston, 3rd edition. ISBN: 9780750675345.

Bordini, R., Hübner, J., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. Wiley. ISBN: 978-0-470-02900-8.

Brandão, F. C., Lima, M. A. T., Pantoja, C. E., Zahn, J., and Viterbo, J. (2021). Engineering approaches for programming agent-based IoT objects using the resource management architecture. 21(23). DOI: 10.3390/s21238110.

Heath, S., editor (2002). *Embedded systems design*. Newnes, Oxford Boston, 2nd ed edition. ISBN: 9780750655460.

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*, CASCON '09, page 122–135, USA. IBM Corp. DOI: 10.1145/1723028.1723044.

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84. DOI: 10.1109/IEEESTD.1990.101064.

Jesus, V., Lazarin, N., Pantoja, C., Manoel, F., Alves, G., Ramos, G., and Filho, J. V. (2022). Proposta de uma ide para desenvolvimento de sma embarcados. In *Proceedings of the 16th Workshop-School on Agents, Environments, and Applications*, pages 49–60, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wesaac.2022.33425.

Kephart, J. and Chess, D. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50. DOI: 10.1109/MC.2003.1160055.

Lazarin, N. and Pantoja, C. (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*, pages 13–20, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wesaac.2015.33308.

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. DOI: 10.1109/ACCESS.2018.2879615.

Pantoja, C. E., Jesus, V. S. d., Lazarin, N. M., and Viterbo, J. (2023). A Spin-off Version of Jason for IoT and Embedded Multi-Agent Systems. In Naldi, M. C. and Bianchi, R. A. C., editors, *Intelligent Systems*, pages 382–396, Cham. Springer Nature Switzerland. DOI: 10.1007/978-3-031-45368-7_25.

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. DOI: 10.1007/978-3-319-50983-9_8.

Ricci, A., Santi, A., and Piunti, M. (2012). Action and Perception in Agent Programming Languages: From Exogenous to Endogenous Environments. In Collier, R., Dix, J., and Novák, P., editors, *Programming Multi-Agent Systems*, pages 119–138, Berlin, Heidelberg. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-28939-2_7.

Russi, D. F. and Charão, A. S. (2011). Ambientes de Desenvolvimento Integrado no Apoio ao Ensino da Linguagem de Programação Haskell. *RENOTE*, 9(2). DOI: 10.22456/1679-1916.25077.

Schimuneck, T. (2014). *Ambiente de desenvolvimento integrado para ensino e programação de microcontroladores da família MCS51*. Monografia (Graduação em Engenharia da Computação), Universidade do Vale do Taquari - Univates, Lajeado. Disponível em: `http://hdl.handle.net/10737/385`.

Siqueira, E., Ramos, G., Raboni, T., Pantoja, C., and Lazarin, N. (2024). Análise comparativa de um protótipo de ide para desenvolvimento de sma embarcados. In *Proceedings of the 18th Workshop-School on Agents, Environments, and Applications*, pages 85–95, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wesaac.2024.33458.

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: 10.22456/2175-2745.110837.

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: 10.1007/978-3-031-37616-0_29.

Wooldridge, M. (2002). Intelligent agents: The key concepts. In Mařík, V., Štěpánková, O., Krautwurmová, H., and Luck, M., editors, *Multi-Agent Systems and Applications II*, pages 3–43, Berlin, Heidelberg. Springer Berlin Heidelberg. DOI: 10.1007/3-540-45982-0_1.

Wooldridge, M. J. (2009). *An introduction to multiagent systems*. Wiley, Chichester, 2. ed. edition. ISBN: 9780470519462.