Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

LFD - MPI: Uma Ferramenta para Desenvolvimento de Aplicações Paralelas Utilizando MPI

Trabalho de Graduação.

Aluno: Rafael Rocha da Silva.

Orientador: Nelson Souto Rosa

Recife,
Julho de 2015

Universidade Federal de Pernambuco
Centro de Informática

Rafael Rocha da Silva

LFD - MPI: Uma Ferramenta para Desenvolvimento de Aplicações Paralelas Utilizando MPI

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Nelson Souto Rosa*

Recife,
Julho de 2015

Dedico este trabalho aos meus pais e irmão,
que sempre fizeram seu melhor por mim. Seu constante
apoio me permitiu superar as diversas etapas da minha vida.

Agradecimentos

Primeiramente, agradeço a Deus, por sua misericórdia e bênçãos em minha vida, que sua vontade seja feita hoje e sempre no meu viver, pois, Senhor, a verdadeira sabedoria do homem está em guardar a tua palavra.

À minha família, principalmente aos meus pais, minha mãe Maria do Socorro e meu pai Cléscio Albino, que sempre se esforçaram para que eu tivesse uma educação de qualidade, por me ensinar a ser um homem temente a Deus e com princípios.

Aos amigos que conheci durante o curso de graduação. Nunca me esquecerei de vocês e dos momentos divertidos que tivemos durante esses anos.

Ao Professor Alan Wagner, e em especial ao Professor e Orientador Dr. Nelson Souto Rosa, por sua dedicação nessa nobre profissão, por ter se disponibilizado e ter me dado a melhor assistência na realização deste trabalho de graduação; por ter sido um excelente professor durante todas as cadeiras que eu tive a oportunidade de ser seu aluno. Serei sempre grato. Muito obrigado por tudo, que Deus lhe abençoe.

Resumo

Este trabalho tem como objetivo expor o a análise e projeto de uma ferramenta de apoio ao desenvolvimento de aplicações paralelas baseadas no padrão MPI (Message-Passing Interface). A ferramenta proposta, chamada LFD-MPI (*Lighweight Formal Development in MPI*), será formulada de maneira a auxiliar a implementação de aplicações MPI com o apoio de métodos formais e geração automática de código. As técnicas formais serão utilizadas na verificação de propriedades desejáveis das aplicações MPI, como o deadlock.

Palavras-chave: MPI**,** aplicações paralelas, deadlock.

Sumário

Capítulo 1
Introdução

A introdução de computadores no meio científico tornou-se um importante método para análise e simulação de fenômenos naturais. Novas soluções de TI são propostas diariamente através de pesquisas e combinação de informações nos diferentes ramos do conhecimento e, dessa forma, a natureza dos problemas tende a ser cada vez mais complexa. Neste contexto, ressaltamos a necessidade de maior poder computacional pelas aplicações atuais.

Tradicionalmente softwares são desenvolvidos para serem executados sequencialmente, ou seja, um problema é dividido em um conjunto de instruções que serão executadas uma após a outra em um certo processador [5], e apenas uma delas é executada em um dado momento de tempo. Porém, o mundo real é paralelo, diversos eventos ocorrem ao mesmo tempo, e esse comportamento precisa ser refletido nos computadores. A partir dessa ideia, o conceito de processamento sequêncial foi estendido dando início ao desenvolvimento de novas arquiteturas de hardware [3, 6] onde processos podem estar sendo executados em diversos processadores ou estarem compartilhando tempo de processamento em um deles.

Porém o Software também precisa passar por mudanças para facilitar e usar esses novos recursos de hardware. Em junho de 1994 foi apresentada a versão 1.0 do MPI (*Message Passing Interface*), um sistema padronizado, direcionado a todos aqueles que querem desenvolver programas paralelos, eficientes e portáveis em Fortran 77 ou C, utilizando o paradigma de troca de mensagens [7]. Seu objetivo era juntar as melhores funcionalidades de cada sistema que utilizavam este paradigma em uma só notação, ao invés de adotar um sistema existente como padrão. Atualmente, um grupo muito importante de programas paralelos são implementados usando MPI [9] e, como mencionado em [8], foi praticamente adotado em todas as áreas científicas que exigem processamento paralelo.

Apesar de sua popularidade, o desenvolvimento de aplicações seguras usando MPI é uma tarefa complexa pois elas têm potencialmente um grande número de erros relacionados a utilização de argumentos inválidos, uso errado de recursos MPI e impasse de mensagens. A identificação de alguns destes erros é um desafio mesmo considerando aplicações simples, porque MPI fornece poucas informações que ajudam a reforçar seu uso correto.

Várias soluções têm sido desenvolvidas para detectar erros MPI [10,11]. Estas soluções tipicamente interceptam chamadas realizadas pela aplicação e, em seguida, verificam a sua corretude. Elas costumam verificar comportamentos indesejados do aplicativo, através de sua execução. Embora esta estratégia seja útil, ela sofre pelo fato de erros potenciais só serem identificados quando o sistema estiver totalmente implementado.

Este trabalho apresentará a análise e projeto iniciais da ferramenta proposta em [1], uma estratégia formal e leve que ajude desenvolvedores de aplicações MPI, desde os estágios iniciais do desenvolvimento.

A estrutura do trabalho está dividida da seguinte maneira:

- Capítulo 1 - Introdução: Apresentará a motivação do trabalho.
- Capítulo 2 - Conceitos Básicos: Este capítulo apresenta o leitor aos conceitos básico para o entendimento do trabalho.
- Capítulo 3 - Descrição da Solução: Neste capítulo é apresentada a solução proposta em [1] para os problemas relacionados com o desenvolvimento em MPI, e que é o objeto de estudo deste trabalho.
- Capítulo 4 - LFD-MPI: o processo de desenvolvimento, Levantamento dos requisitos, Casos de Uso, Análise, Projeto e Tecnologias utilizadas na implementação da ferramenta.
- Capítulo 5 - Utilização da ferramenta: Manual de utilização da ferramenta propsta.
- Capítulo 6 - Conclusão: As considerações finais são apresentadas.
- Referências Bibliográficas.

Capítulo 2
Conceitos Básicos

A seguir serão apresentados conceitos necessários ao entendimento da proposta deste trabalho. Inicialmente abordaremos MPI, em seguida, a linguagem e tecnologia utilizadas na verificação formal das aplicações desenvolvidas com a ferramenta LFD-MPI. Vale salientar que estamos interessados em apenas uma propriedade formal, que é a presença ou não de deadlock.

2.1 Message Passing Interface (MPI)

MPI (Message Passing Interface) [2] é uma API padrão para programação paralela. MPI tem sido adotado como padrão industrial [12] e tem sido amplamente utilizado em plataformas HPC (High Performance Computing).

Entre as vantagens na utilização de MPI destacam-se:

- MPI é a especificação padrão no desenvolvimento de programas que manipulam bibliotecas de troca de mensagens;
- A versão MPI-3 possui mais de 430 primitivas o que permite que desenvolvedores de aplicações paralelas tenham recursos para implementar diferentes tipos de funcionalidades;
- existe uma grande variedade de implementações no mercado, tanto comerciais quanto públicas; e
- ela é portável, o código desenvolvido para uma plataforma pode ser passado para outra plataforma que suporte a especificação MPI com poucas alterações (ou nenhuma, dependendo do caso).

A estrutura de um programa CSP segue o modelo apresentado na figura 1. Podemos ainda classificar as principais primitivas MPI em 3 grupos:

- **Rotinas de gerenciamento de ambiente:** Utilizadas para obter e definir informações do ambiente de execução, por exemplo: MPI_Init e MPI_Finalize.
- **Rotinas de comunicação ponto a ponto:** Utilizadas para a troca de mensagens entre processos. Elas podem ser bloqueantes (MPI_Send, MPI_Recv, etc) ou não-bloqueantes(MPI_IRecv, MPI_Wait, etc).

- **Rotinas de comunicação coletiva:** Utilizadas para comunicação coletiva dentro de um grupo (MPI_Barrier,etc).

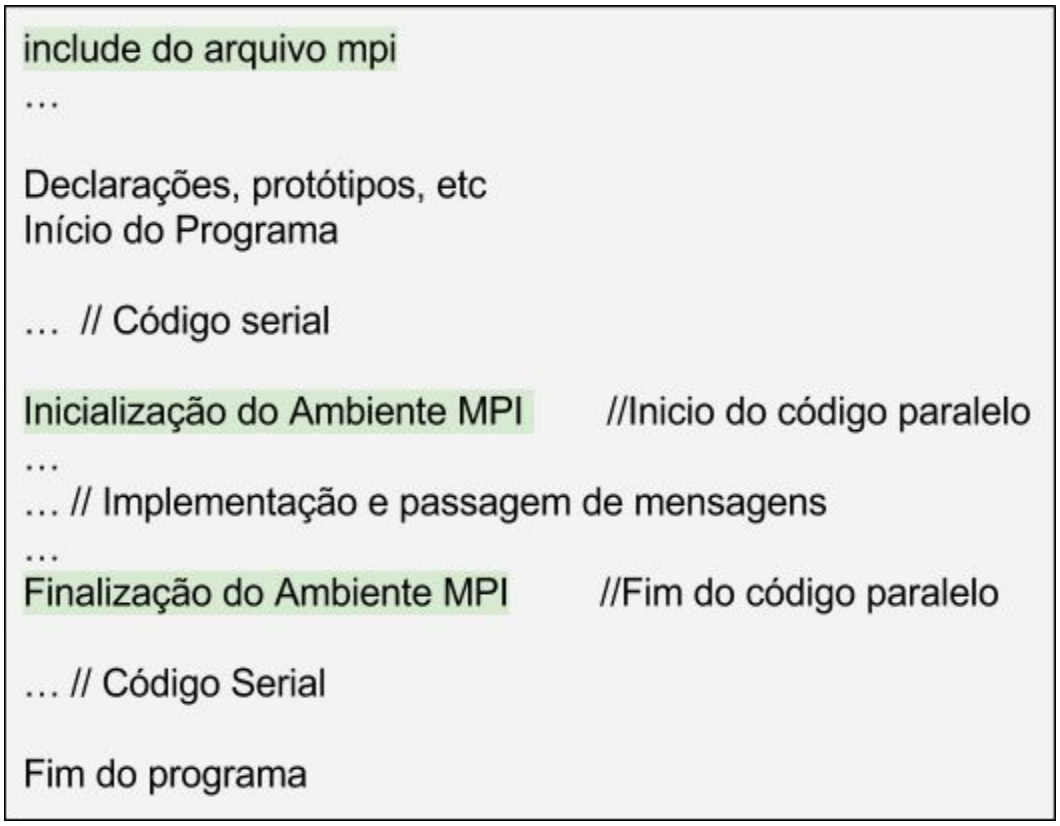

Figura 1. Estrutura de código MPI.

2.2 Communicating Sequential Processes (CSP)

CSP é uma álgebra de processo [4] utilizada para descrever sistemas concorrentes. Ela é formada por um conjunto de modelos matemáticos utilizados para descrever e analisar sistemas onde diferentes componentes interagem a nível de comunicação. O interesse de CSP não é descrever o que acontece em cada ciclo de clock e sim descrever o comportamento de, por exemplo, um algoritmo descrito nessa notação.

CSP tem suas origens na segunda metade da década de 70, uma época onde os principais problemas envolvendo processamento paralelo surgiam de áreas como desenvolvimento de sistemas operacionais e multitasking [13]. Atualmente, os vários problemas estão relacionados à distribuição de tarefas entre os diferentes processadores, o que se feito incorretamente gera gargalos no desempenho da aplicação.

CSP apresenta versões Timed e Untimed, a comuniação handshake é um exemplo de CSP untimed, onde temos a abstração dos eventos ao longo do tempo. A versão timed é necessária onde sistemas precisam se apoiar em uma demostração concreta dos processos que ocorrem ao longo do tempo, para garantir a corretude do sistema e além de servir como embasamento para novas teorias de sistemas untimed.

Um exemplo bastante comum utilizado para indrodução dos leitores aos conceitos da descrição CSP é o da máquina ATM, porque o grande número de transações e de máquinas que precisam operar simultaneamente, além de interagirem com um banco central de dados, representam um cenário ideal para a criação de uma modelagem formal. A seguir será dada uma breve introdução desse exemplo, ilustrando apenas a interface entre o usuário e a máquina.

Cenário: “Interface entre máquina ATM e usuário”

O primeiro passo da modelagem utilizando CSP (e o mais importante), é a definição de todos os eventos de comunicação envolvidos no ambiente. Esses eventos irão fazer parte de um alfabeto Σ contendo todas as comunicações entre os processos envolvidos. Quanto maior a quantidade de eventos levantados na modelagem, maior será o nível de detalhamento da especificação. No nosso exemplo foram levantado os eventos: in.c, out.c (entrada e saida de cartões) para todos os cartões $c \in CARD$ (pertencente ao grupo de cartões CARD, reconhecidos pela máquina); pin.p todos os possíveis números PIN representados por p; e por último, “req.n” , “dispense.n” ou “refuse” para todos os valores de saque “n” possíveis (pertencentes ao conjunto WA, Withdrawal Amount, em português, valor ou quantidade do saque).

O próximo passo é modelar as ações ocorridas. Neste exemplo, a máquina irá seguir o seguinte passo a passo: Aceitar o cartão (inseção do cartão na máquina), solicitar o número PIN do cartão, fornecer a quantidade de dinheiro solicitada, e por fim, retornar o cartão ao usuário. Vale lembrar que estamos modelando um cenário de sucesso, não estamos levando em conta que o número PIN, cartão ou valor do saque fornecidos à máquina são inválidos. Todo o processo é descrito de acordo com a seguinte equação:

$$\mathbf{ATM\ 1 = in?c : CARD \rightarrow pin.fpin(c) \rightarrow req?n : WA \rightarrow dispense!n \rightarrow out.c \rightarrow ATM\ 1}$$

O processo descrito como ATM1 na equação acime é formado por diversas partes: “in?c : CARD”, “pin.fpin(c)”, “req?n : WA”, “dispense!n” e “out.c”; todos separados por uma seta “→”. A primeira notação que devemos ter em mente é a seta “→”; ela é utilizada na operação de prefixação, por exemplo:

$$P = a \rightarrow STOP$$

O processo P (representados por letras maiúscula), pode ser visto como um um processo que espera um evento “a” acontecer e vai para o processo STOP (é o processo mais simples em CSP, o qual não faz nada), a utilização do operador “→” prefixa os processos, dando a ideia de sequência.

Analisando a expressão “in?c : CARD” introduzimos outro conceito, que são os “canais” em CSP. É esperado que processos em CSP sejam capazes de receber inputs e liberar outputs, logo a expressão “in?c : CARD” representa a seguinte notação “canal?elemento :TIPO” e pode ser traduzida da seguinte maneira: o canal “in” do tipo CARD irá receber (o operador “?” indica input), um elemento “c” (espera-se que esse elemento faça parte do conjunto CARD).

A próxima expressão “pin.fpin(c)” tem o seguinte significado: “fpin(c)” é a função que irá verificar a corretude do “pin” para o cartão “c”. A máquina então solicita que o usuário insira a quantidade que ele quer retirar atráves da expressão “req?n : WA”, como visto anteriromente, pode ser traduzida em: canal “req”(requisição) do tipo WA (withdraw amount ou valor do saque), irá receber um valor “n”.

Continuando o processo temos a expressão “dispense!n” que representa “canal!elemento” e pode ser traduzida da seguinte forma o canal “dispense”, irá fornecer (o operador “!” significa output), o elemento “n”, que no caso é a quantidade solicitada. Continuando temos a expressão “out.c” este é o operador de composição representado: o evento denominado “out” do elemento c, ou seja, a devolução do cartão ao usuário. E por fim, a expressão “ATM1” indica a repetição do processo, ou seja, a máquina irá voltar para a fase inicial e estará pronta para realizar todas as operações novamente.

Apenas com este exemplo podemos notar que lidar com as notações de CSP, pode ser uma tarefa complexa (foram apresentados apenas alguns dos operadores mais básicos da linguagem). Neste exemplo, consideramos apenas o fluxo onde todos os processos ocorrem sem erro, porém, outras aplicações/exemplos, necessitam que todos os cenários sejam abordados. Por exemplo, o simples fato de o usuário esquecer a sua senha (PIN), poderia causar deadlock na máquina com o cartão ainda dentro dela. Logo, as modelagem precisam ser feitas cuidadosamente.

2.3 Failures/Divergences Refinement (FDR)

FDR[1] é uma ferramenta para verificar propriedades em programas escritos em CSP, ou mais especificamente, CSPM (machine-readable CSP) que é uma linguagem funcional com suporte às definições de processos na notação CSP, e que permite que

[1] https://www.cs.ox.ac.uk/projects/fdr/manual/

o programador estabeleça as chamadas “assertions” que são tags indicando quais as propriedades que a ferramenta FDR irá analisar.

FDR 3 é disponível para Linux, MAC e Windows em 2 formas: “A FDR3 Tool” que é um programa gráfico capaz de carregar arquivos CSPM, checar “assertions” sobre os processos e fornecer um robusto debbuger que permite ao usuário verificar o motivo de sua falha; e uma ferramenta em linha de comando que também permite a verificação de assertions e pode ser integrada com outras ferramentas, mas que não possui a mesma quantidades de recursos para o entendimento como a ferramenta principal, a saber, a navegação pelos “contra-exemplos“.

A versão mais completa de ferramenta é formada por: um Prompt interativo, um campo Assertions, e um campo Tasks como mostrado na figura 2.

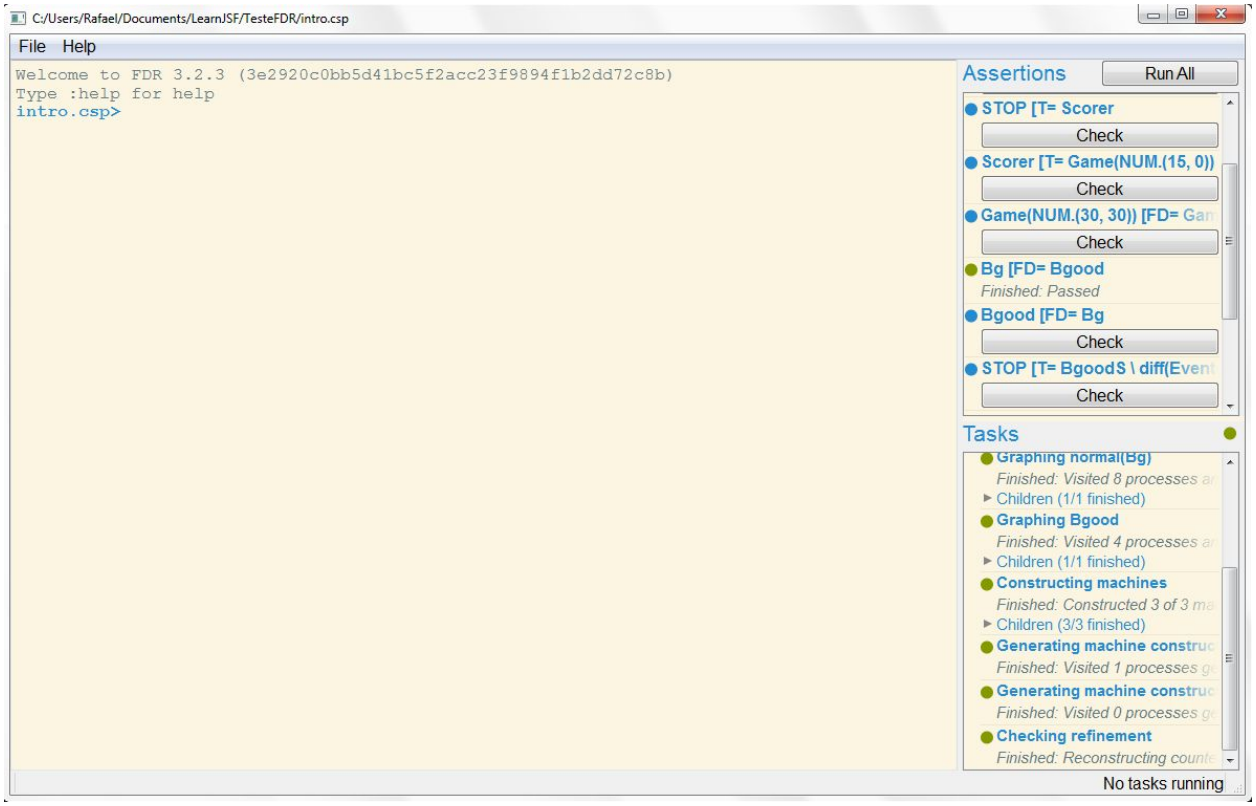

Figura 2: Sessão da Ferramenta FDR.

- **Prompt:** O prompt da ferramenta permite que o usuário, carregue arquivos “.csp”, navegue pelos diretórios, altere opções de processamento, explore as transições dos processos, etc. Além disso o prompt permite que o usuário escreva declarações csp e execute-as.
- **Assertions:** A tela “Assertions”, como o próprio nome indica, mostra a lista de assertions definidas pelo usuário no arquivo “.csp” carregado na ferramenta. As

assertions podem ser executadas uma a uma clicando no respectivo botão “check”, ou todas ao mesmo tempo no botão “Run All”. As assertions que passaram com resultado positivo, são marcadas com um indicador verde; as que falharam são marcadas com um indicador vermelho, as que não foram verificadas ainda, permanecem em azul e as que estão sendo verificadas no momento recebem um indicador amarelo.

- **Tasks:** A tela Tasks mantém o histórico das ações que foram ou estão sendo executas. Ela também mostra informações de refinamentos de verificação, validação de expressões, etc.

Todas as funcionalidades da ferramenta também são oferecidas na forma de uma biblioteca “libfdr” 64 bits disponível para C++, Java e Python. Tomando como exemplo a versão Java (utilizada neste trabalho), para utilização dos recursos FDR, são necessários 2 passos:

- primeiro, importar o arquivo “.jar” e suas dependencias para o projeto (Web, Desktop, etc).
- Segundo, deve ser importado o pacote “uk.ac.ox.cs.fdr.NOME_DA_CLASSE” se o usuário quiser usar uma classe especifica ou “uk.ac.ox.cs.fdr.*” se o usuário quiser importar para o projeto, todas as classes do FDR (o asterístico final indica a importação de todas as classes desse pacote).

Após a realização desses dois passos, o usuário poderá utilizar todas as classes (e consequentemente seus métodos), por exemplo, a criação de uma nova sessão FDR o carregamento de um arquivo “.csp” para esta sessão pode ser feita da seguinte forma:

```
1  Session session = new Session();
2  session.loadFile("meuArquivo.csp");
```

As classes e métodos do FDR podem ser encontradas na documentação dísponivel na página oficial do FDR 3.

Capítulo 3
Descrição da Solução

Como mencionado no capítulo 1 e constatado em [8, 9], MPI é amplamente utilizado no desenvolvimento de aplicações que utilizam o paradigma de passagem de mensagens, porém, vários erros relacionados à utilização de argumentos inválidos, uso errado de recursos MPI e impasse de mensagens, dificultam até mesmo a realização de aplicações simples, além de pouca informação que MPI fornece para reforçar seu uso correto.

Este problemas motivaram a solução apresentada em [1], baseada em 3 aspectos: (i) um processo de desenvolvimento, (ii) um modelo formal para especificar os programas durante o desenvolvimento, e (iii) uma ferramenta que suporta as etapas do processo proposto. Este trabalho concentra-se no terceiro item, ou seja, o desenvolvimento de uma ferramenta de suporte ao processo de desenvolvimento.

3.1 Processo de desenvolvimento

O processo de desenvolvimento de uma nova aplicação MPI, na ferramenta LFD-MPI é composta por 4 fases: design estrutural, design de comunicação, formalização e mapeamento de implementação.

Na primeira fase são adicionados os componentes que formam a aplicação. Será criada uma representação visual da aplicação MPI, utilizando-se figuras geométricas e ligações que representam um ambiente onde diversos processos interagem uns com os outros. Essa forma de representação visual auxilia o entendimento do programa MPI, pois o usário terá uma forma de visualizar o que seu código irá representar (nesta fase ainda não estamos preocupados em programação). O artefato gerado nesta fase chama-se “especificação abstrata” e ele é a modelagem visual do programa MPI que está sendo desenvolvido.

A segunda fase é o momento onde o usuário irá utilizar primitivas MPI para definir as interações entre esses componentes, ou seja, nesta etapa o usuário irá criar a semântica da sua aplicação. Cada elemento adicionado na fase de design estrutural, irá receber um conjunto de primitivas que irá modelar o seu comportamento com outros elementos e no ambiente da aplicação como um todo. Ainda nesta fase, serão realizadas algumas operações, por exemplo, a verificação sintática das primitivas utilizadas para garantir a corretude da aplicação, porém esse assunto será abordado

nos próximos capitulos. O artefato desta fase é a especificação concreta; ela é formada a partir da especificação abstrata (representação visual criada no passo anterior) mais a adição do comportamento de cada elemento atráves das primitivas MPI.

A terceira fase, chamada de formalização, é composta por duas etapas: o mapeamento formal e a verificação. O mapeamento formal é a operação de criação de um programa CSP a partir da especificação concreta (artefato da fase anterior) que será utilizado como entrada para um módulo na etapa de verificação. Estamos interessados apenas na verificação de uma propriedade formal, " presença de deadlock", para isso iremos utilizar uma ferramenta externa que dá suporte a esta funcionalidade. Se a propriedade não for corretamente satisfeita, ou seja não está livre de deadlock, o usuário será informado do erro e consequentemente precisará corrigir algum ponto de sua modelagem estrutural e/ou de comunicação. O programa CSP, criado e verificado, é o artefato da terceira fase, e o chamamos de "especificação formal".

Por fim, a quarta fase, chamada de mapeamento de implementação, é responsável por construír um esqueleto de aplicação MPI (em C), que serve como base para a implementação real do sistema. Assim como a formalização, esta operação também será feita sobre a especificação concreta, ou seja, se programa o CSP representando a aplicação estiver livre de deadlock, é seguro criar um protótipo de programa em MPI. Dessa forma, satisfazemos o principal objetivo da ferramenta que é: criar código MPI, com verificação de deadlock desde os primeiros passos da implementação. O código MPI criado é denominado "esqueleto de implementação", e é o artefato da quarta e ultima fase.

A figura 3 ilustra o processo de desenvolvimento assim como os artefatos produzidos em cada uma das etapas. A especificação abstrata, como mencionado anteriormente, é a designação dada à estrutura que descreve a aplicação, ou seja, o conjunto de componentes visuais criados pelo usuário. A especificação concreta é o enriquecimento da especificação abstrata com as definições de comunicação (adicionadas durante a fase de design de comunicação). A especificação formal é a especificação concreta após a validação da propriedade desejada (deadlock-free), nesta fase, a especificação concreta é transformada em um código descrito em uma linguagem de álgebra de processos, que no caso deste trabalho, é a CSP (existem diversas linguagens pertencentes a esta família, por exemplo LOTOS). E finalmente, é gerada um esqueleto da implementação do projeto em MPI utlilizando C.

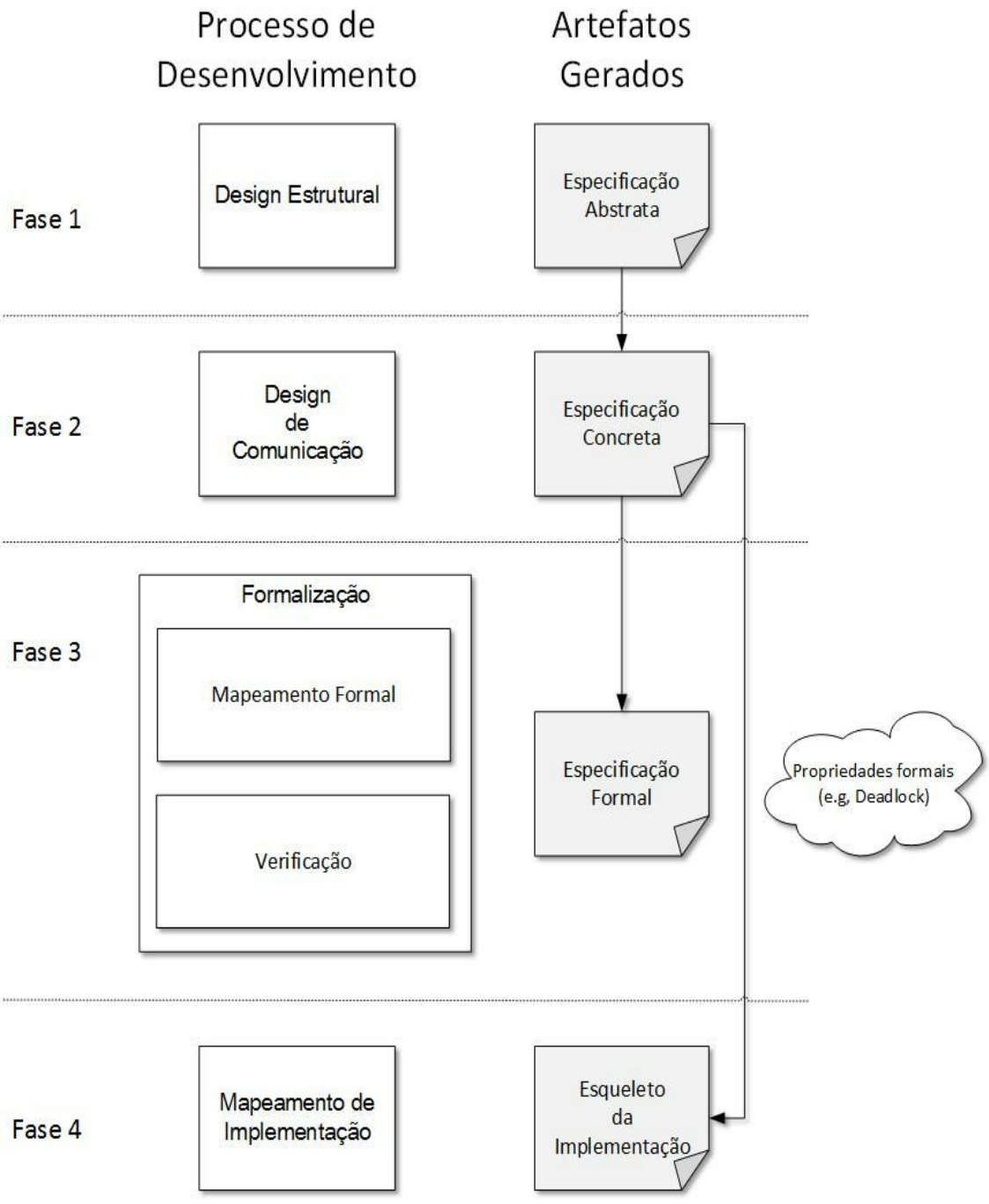

Figura 3. Visão geral do processo de desenvolvimento.

3.2 A ferramenta

A ferramenta proposta em [1] é chamada de LFD-MPI (Lightweight Formal Development in MPI) e é projetada para suportar as etapas: design estrutural (fase 1), design de comunicação (fase 2), mapeamento de implementação, e da formalização ela será responsável apenas pela parte de geração do código CSP (a etapa de mapeamento formal da fase 3). A verificação de deadlock será feita por uma ferramenta externa chamada “FDR 3” (Failures and Divergences Refinement) que é especializada na checagem de propriedades formais de programas escritos em CSP. A seguir serão explicados cada uma dos componentes e seu papel dentro da ferramenta.

Como mostrado na figura 4, LFD-MPI é composta por 4 principais módulos: Um editor gráfico, dois componentes de mapeamento: “C2F” e o outro “C2I”, e por fim, um adaptador formal.

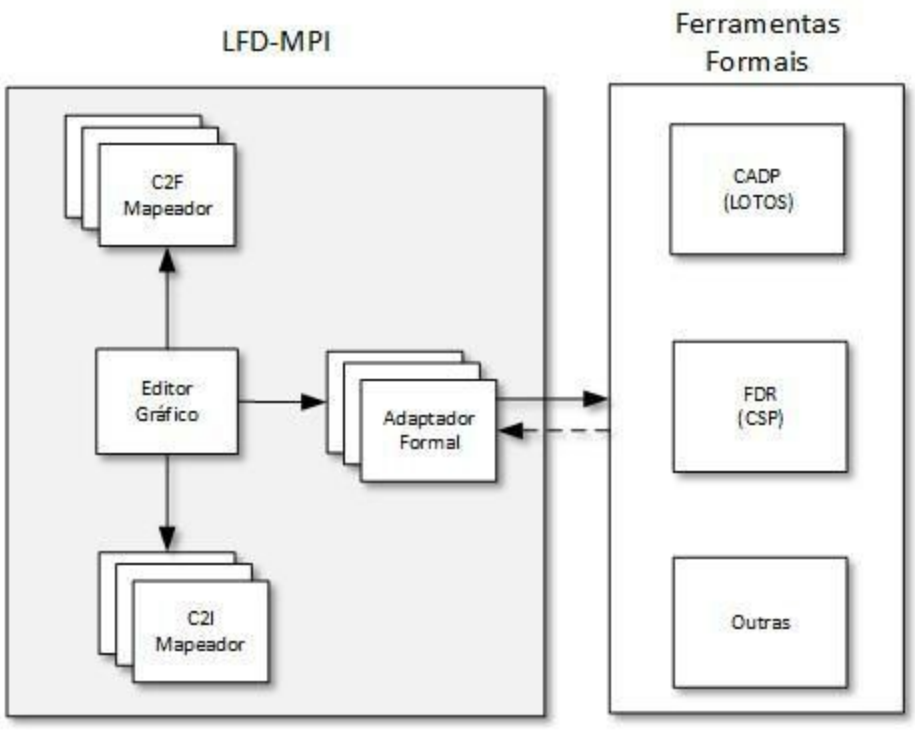

Figura 4. Arquitetura do LFD-MPI

3.2.1 Editor Gráfico:

O editor gráfico é a parte central da ferramenta. Ele fornece elementos gráficos e áreas de texto, que serão utilizados para representar a estrutura da aplicação e o comportamento de cada um dos componentes adicionados nessas duas primeiras fases.

A ideia é que este editor tenha um conjunto de figuras que possam ser escolhidas e então utilizadas para representar os diversos processos de uma aplicação MPI. A ferramenta terá um canvas onde essas figuras poderão ser adicionadas através de um clique e também será possível alterar propriedades desses elementos, entre elas, o seu comportamento (utilizando primitivas MPI). É esperado que esse editor seja de fácil manuseio como os diferentes tipos de programas de modelagem gráfica existentes, a exemplo, Microsoft Visio, Bizagi, Cacco, entre outros, salvo suas respectivas finalidades e funcionalidades.

3.2.2 Mapeadores “C2F” e “C2I”:

Os dois módulos de mapeamento serão responsáveis por prover as transformações de código na etapa de mapeamento formal (fase 3) e na fase de mapeamento de implementação.

Na fase de mapeamento formal, o mapeador C2F (Concrete to Formal) será chamado para criar um código CSP a partir da especificação concreta. Este código será a entrada para a ferramenta FDR3, a qual irá dizer se o projeto criado, apresenta ou não deadlock. As regras de mapeamento para criação de um código CSP ainda estão em processo de definição, portanto, vamos assumir que este módulo já está implementado e funcionando no decorrer do projeto.

Na fase de mapeamento de implementação precisamos gerar o código MPI. É neste momento que o mapeador “C2I” será ativado, para criar um código em C descrito em MPI. As regras de mapeamento para MPI também estão em processso de definição, dessa forma, também vamos assumir que a operação de conversão da especificação concreta para a o esqueleto de implementação já está implementada.

3.2.3 Adaptador Formal:

Como apresentado, a única funcionalidade não fornecida pela ferramenta LFD-MPI é a verificação de deadlock. Ela precisará se comunicação com uma ferramenta externa e então solicitar essa operação.

O adaptador formal é interface de comunicação entre a ferramenta LFD-MPI e os programas formais. Na fase de verificação, a especificação formal (código CSP) será analisada por uma ferramenta externa responsável por verificar as propriedades formais do código. É neste momento que o adaptador formal será responsável pela multiplexação das invocações aos métodos da ferramenta formal (de acordo com a linguagem escolhida), por exemplo, se a especificação formal for descrita em LOTOS, o adaptador irá chamar métodos de verificação da ferramenta CADP, específica para a linguagem LOTOS. Como a utilização de diferentes ferramentas formais está além do

escopo deste trabalho (esta funcionalidade é considerada um projeto futuro à ferramenta), será considerado apenas o desenvolvimento com CSP, e consequentemente a verificação com FDR.

Observa-se que a ferramenta não é acoplada à um tipo especifico de descrição. Tanto os mapeadores quanto os adaptadores formais podem ser desenvolvidos de maneira a abranger uma variedade de linguagens. É possível termos diferentes tipos de mapeadores, um para cada linguagem, por exemplo, um mapeador para ACP (Bergstra & Klop's *Algebra of Communicating Processes*), outro para CCS (Milner's *Calculus of Communicating Systems*) e os adaptadores para as respectivas ferramentas de verificação de cada uma dessas linguagens, de acordo com as decisões de projeto tomadas pelo desenvolvedor.

O próximo capitulo serão referentes ao processo de levantamento de requisitos, casos de uso, análise, projeto e Implementação da ferramenta.

Capítulo 4
LFD-MPI

Este capítulo irá apresentar os passos desde o levantamento de requisitos às decisões adotadas na implementação da ferramenta LFD-MPI.

4.1 Requisitos

O levantamento de requisitos é um passo muito importante na elaboração de um novo projeto de software. Cerca de 80% do tempo de desenvolvimento de uma nova ferramenta é gasto na correção de erros originados por um mal levantamento de seus requisitos [16]. Além disso, o custo relacionado à correção desses erros, cresce quanto mais tardio for detectado. Por isso, esta seção do trabalho é dedicada a mostrar o levantamento de requisitos iniciais da ferramenta LFD-MPI.

Cada requisito é identificado por um código único e um nome, além de contar com uma breve descrição, indicação dos casos de uso que implementam a funcionalidade descrita e uma prioridade, que pode ser:

- **Essencial:** Se o software for considerado inaceitável quando o requisito não for atendido da maneira acordada.
- **Desejável:** Se o requisito possibilitar melhorias significativas no sistema, mas não torná-lo inaceitável caso não seja implementado.
- **Opcional** Se o requisito possibilitar apenas melhorias moderadas no sistema, devendo ser atendido após a conclusão de todos os outros, caso o cliente julgue conveniente.

4.1.1 Requisitos Funcionais

Requisitos funcionais definem as funcionalidades que o sistema deve implementar [14, 15]. A descrição da ferramenta apresentada nas seções 3.1 e 3.2, evidenciam alguns pontos importantes que devem ser levados em consideração nesta primeira análise das atividades da aplicação. A seguir estão descritos os requisitos funcionais levantados e validados até o momento.

RF[001] Gerar Código MPI	
Descrição	Gerar um código simples em MPI. Por se tratar de uma versão inicial do programa, iremos abordar o mínimo de primitivas MPI possível, para manter o desenvolvimento simples nesta fase.
Prioridade	Essencial

RF[002] Gerar Especificação Formal	
Descrição	A ferramenta deve ser capaz de gerar código CSP (especificação formal), para que FDR possa fazer a verificação de deadlock.
Prioridade	Essencial

RF[003] Verificação de Deadlock	
Descrição	Realizar a verificação da existência de deadlock no programa desenvolvido, utilizando métodos formais fornecidos por uma ferramenta externa.
Prioridade	Essencial

RF[004] Verificação Sintaxe	
Descrição	Verificação da sintaxe dos processos de acordo com a notação estabelecida.
Prioridade	Essencial

RF[005] Salvar XML	
Descrição	Gerar um arquivo XML contendo as informações do programa escrito com a ferramenta.

Prioridade	Essencial

RF[006] Carregar XML	
Descrição	Carregar um arquivo XML contendo informações de um programa escrito com a ferramenta
Prioridade	Essencial

4.1.2 Requisitos não funcionais

Requisitos não-funcionais definem as restrições ao desenvolvimento e projeto do sistema, tais como aspectos de desempenho, interfaces com o usuário, confiabilidade, segurança, manutenibilidade, portabilidade e padrões de projeto [14].

4.1.2.1 Requisitos de produto:

Os requisitos de produto definidos abaixo, definem às características desejadas para o produto a primeira versão da ferramenta LFD-MPI.

RNF[001] Facilidade de Aprender	
Descrição	A utilização do sistema deve ser intuitiva e de fácil aprendizagem. Os usuários devem se adaptar ao uso do sistema sem muita dificuldade e receber suporte para executar funções com as quais não estiver habituado.
Prioridade	Essencial

RNF[002] Simplicidade em Usar	
Descrição	O sistema deve ser de utilização objetiva e ágil. As ações devem ser executadas em poucos passos.
Prioridade	Essencial

RNF[003] Interface amigável	
Descrição	A tela visualizada pelos usuários deve ser intuitiva e proporcionar um rápido entendimento da informação exibida.
Prioridade	Essencial

4.1.2.2 Requisitos de processo:

Os requisitos de processo definem as restrições que devem ser impostas ao processo de desenvolvimento do sistema [14,15]. Para esta primeira versão da ferramenta LFD-MPI, foi identificado e validado apenas uma restrição.

RNF[004] Utilização de Framework Java	
Descrição	A API escolhida para desenvolvimento com FDR foi versão Java, por isso é necessário escolher um framework que consiga manusear os arquivos ".jar".
Prioridade	Essencial

4.2 Casos de Uso

Os requisitos iniciais identificados na seção anterior foram convertidos nos casos de usos que se relacionam conforme o modelo apresentado na figura 5. A seguir, cada um dos casos de uso será detalhado de acordo com os seguintes critérios: Identificador, prioridade, pré-condição, pós-condição, fluxo de eventos primário e secundário.

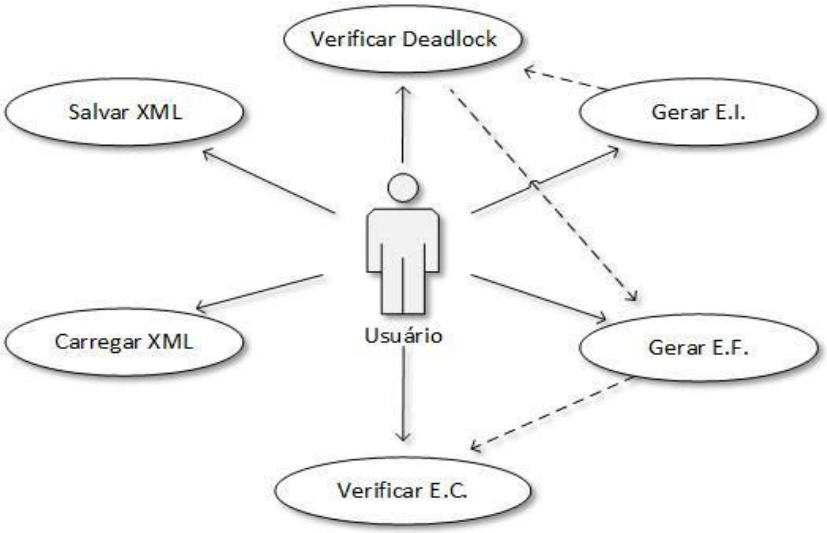

Figura 5. Diagrama dos casos de uso.

4.2.1 Verificação Sintática da Especificação Concreta

Identificador: [UC001].
Prioridade: Média.
Pré-condição: Nenhuma
Pós-condição: O sistema deve informar ao usuário se a especificação concreta está de acordo com a sintaxe definida para representação da mesma.
Fluxo de eventos principal:

1. O sistema captura o texto digitado na área de codificação da especificação concreta de um determinado processo.
2. É analisado linha por linha o texto de acordo com a sintaxe estabelecida.
3. O sistema informa ao usuário se a especificação concreta do processo analisado obedece as regras de sintaxe.

Fluxo de eventos secundário:

1. No passo 2, se houver algum erro sintático, o sistema possivelmente indica ao usuário o tipo de erro.

4.2.2 Gerar Especificação Formal

Identificador: [UC002].
Prioridade: Alta.
Pré-condição: A especificação concreta não deve possuir erros de sintaxe.

Pós-condição: O sistema deve retornar a especificação formal descrita em CSP.
Fluxo de eventos principal:

1. O usuário solicita ao sistema a geração da especificação formal.
2. O mapeador C2F traduz a especificação concreta em especificação formal descrita em CSP.
3. O sistema retorna a especificação formal em uma área de texto reservada.

Fluxo de eventos secundário: Não possui.

4.2.3 Verificar Deadlock

Identificador: [UC003].
Prioridade: Alta.
Pré-condição: Ter gerado a especificação formal.
Pós-condição: Indicar positiva ou negativamente a presença de deadlock na modelagem da aplicação feita pelo usuário.
Fluxo de eventos principal:

1. Usuário solicita ao sistema a verificação de deadlock
2. O adaptador formal é acionado e será encarregado de chamar o respectivo método de verificação da ferramenta formal.
3. Adaptador retorna resultado da checagem para o editor gráfico.
4. Resposta é apresentada ao usuário.

Fluxo de eventos secundário:

1. Se no passo 3 o resultado for positivo para deadolock, o usuário deverá voltar para as etapas de modelagem estruturais e/ou comunicação para correção de erros.

4.2.4 Gerar Esqueleto de Implementação

Identificador: [UC004].
Prioridade: Alta.
Pré-condição: A especificação formal deve ter indicação negativa para deadlock.
Pós-condição: Retornar esqueleto de código MPI em C que será utilizado como ponto de partida para a implementação de fato.
Fluxo de eventos principal:

1. Usuário solicita ao sistema a geração do esqueleto MPI.

2. Sistema aciona mapeador C2I, que será responsável por fazer a tradução da especificação formal para o esqueleto de implementação.
3. mapeador retorna o resutado para o editor gráfico.
4. O resultado é mostrado ao usuário na area de texto reservada para o esqueleto da implementação.

Fluxo de eventos secundário: Não possui.

4.2.5 Salvar Arquivo XML

Identificador: [UC005].
Prioridade: Alta.
Pré-condição: Nenhuma.
Pós-condição: Disponibilizar o download de um arquivo XML que contém a representação da modelagem feita pelo usuário.
Fluxo de eventos principal:

1. Usuário solicita ao sistema salvar o seu projeto.
2. O sistema gera um modelo XML do projeto.
3. A tela de download é gerada; o usuário poderá selecionar o local onde o arquivo será salvo.

Fluxo de eventos secundário: Não possui.

4.2.6 Carregar Arquivo XML

Identificador: [UC006].
Prioridade: Alta.
Pré-condição: Nenhuma.
Pós-condição: O arquivo XML é carregado no sistema, e sua informação é apresentada no editor gráfico.
Fluxo de eventos principal:

1. Usuário seleciona opção de carregamento e arquivo XML no sistema.
2. A tela para a escolha do arquivo é mostrada.
3. Usuário seleciona arquivo de projeto que ele queira carregar.
4. O sistema carrega a informação contida no XML diretamente no editor gráfico.

Fluxo de eventos secundário: Não possui.

4.3 Análise

Nesta seção serão apresentados os passos realizados no processo de análise da ferramenta LFD-MPI. O levantamento dos requisitos e dos casos de uso mostrados nas seções anteriores são as entradas para o processo aqui descrito. Vale salientar que a ferramenta é um projeto em desenvolvimento, logo, o estudo aqui apresentado visa apenas introduzir o leitor à uma versão inicial do programa, abordando apenas os seus módulos mais básicos.

4.3.1 Verificação Sintática da Especificação Concreta:

Usuário solicita a verificação sintática da especificação concreta à "TelaVerificaEC" passando a especificação concreta como parâmetro, esta por sua vez chama o método de verificação "analiseSintatica" do controlador responsável pela verificação (chamado "ControladorEC"). O controlador então retorna o resultado da verificação para a tela, que por fim, mostra o resultado ao usuário.

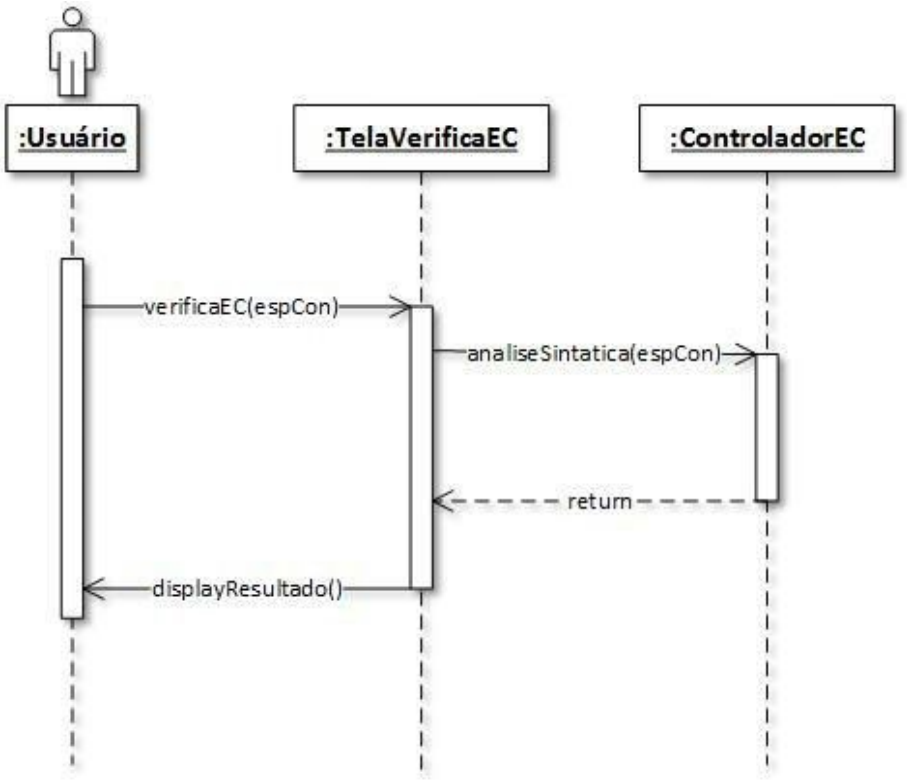

Figura 6. Operação de verificação sintática da especificação concreta.

4.3.2 Gerar Especificação Formal:

Usuário solicita a geração da especificação formal à “TelaGerarEF” passando como parâmetro a especificação concreta, a tela por sua vez, chama o método de geração de especificação formal do “ControladorGerarEF”. O controlador então retorna o resultado da operação para a tela, esta exibe a especificação formal para o usuário conforme ilustrado na figura 7.

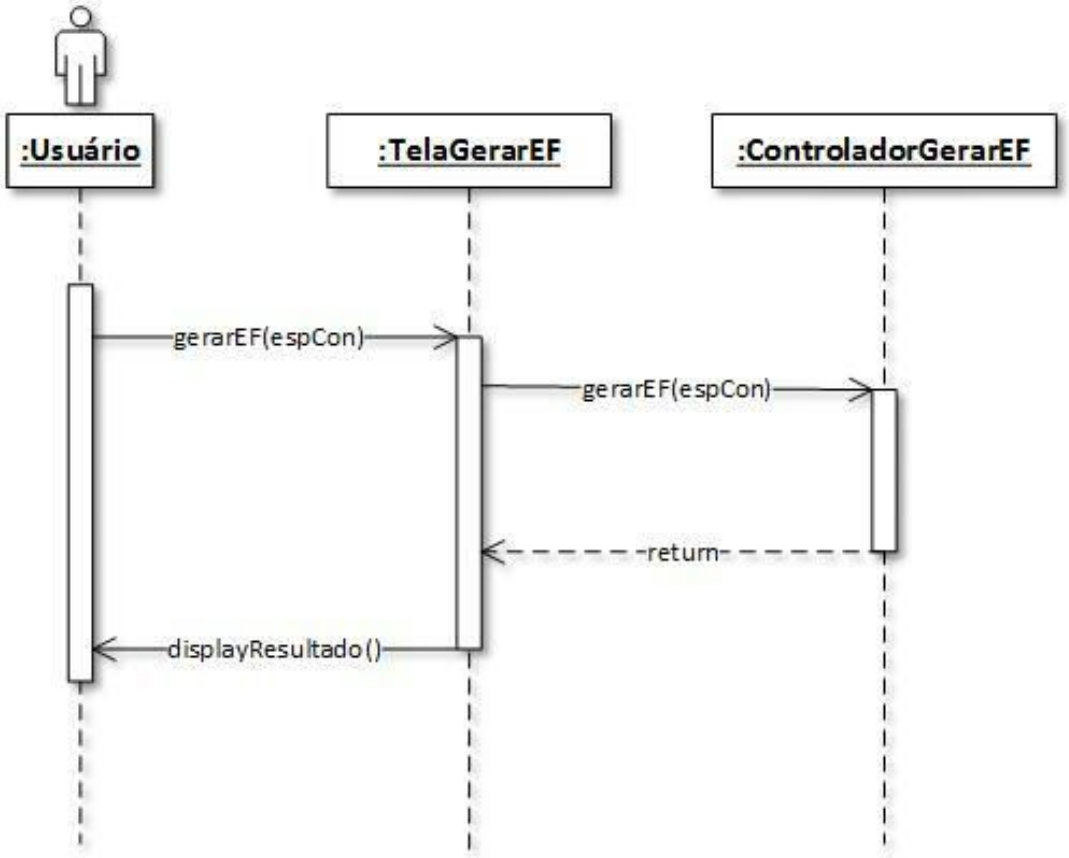

Figura 7. Operação gerar especificação Formal.

4.3.3 Verificar Deadlock:

Usuário solicita a verificação de deadlock à “TelaVerificaDL” (neste momento assume-se que o usuário tenha pelo menos modelado a especificação concreta). Se o usuário ainda não tenha gerado a especificação formal, a tela irá chamar o método de geração formal do controlador “ControladorGerarEF”. Uma vez formada a especificação

formal, ela é passada para o “ControladorVerificarDL” chamando a interface com o FDR (comunicador com um sistema externo). A interface irá então solicitar ao FDR a checagem da propriedade de deadlock. A interface recebe a resposta, encaminha para o controlador, que novamente a encaminha para a tela onde o resultado da checagem será apresentado ao usuário. O processo pode ser observado na figura 8.

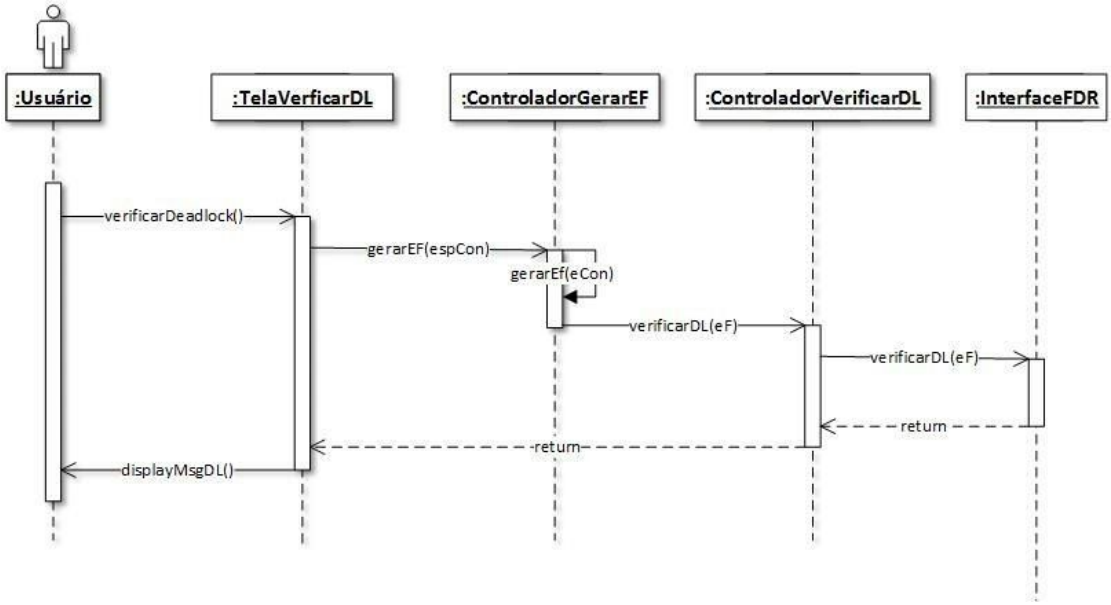

Figura 8. Operação de verificação de deadlock

4.3.4 Gerar Esqueleto de Implementação:

Usuário solicita a geração do esqueleto de implementação à “TelaGerarEI” passando como parâmetro a especificação concreta já verificada contra existência de deadloack, a tela então chama o “ControladorEI” responsável por fazer o mapeamento da especificação concreta em esqueleto de implementação. Terminada a operação, o resultado é retornado à tela onde será exibido para o usuário de acordo com figura 9.

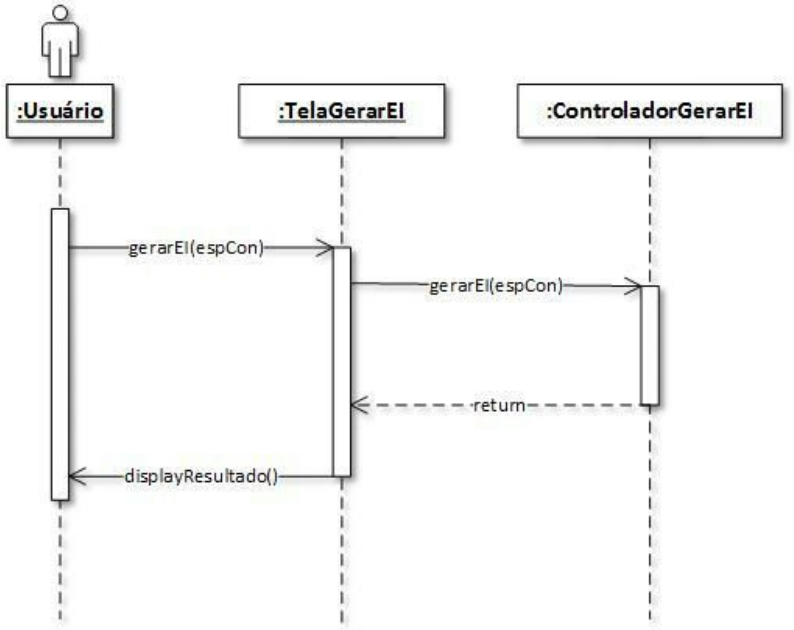

Figura 9. Operação gerar esqueleto de implementação**.**

4.3.5 Salvar Arquivo XML:

Usuário solicita a geração do XML de sua modelagem à “TelaSalvarXml” passando como parâmetro a especificação concreta (modelagem estrutural mais a de comunicação). A tela aciona o “ControladorSalvarXML” que então irá solicitar a seleção do diretório onde o arquivo vai ser salvo (a chamada é passada para a tela e a tela exibe ao usuário). O diretório então é encaminhado ao controlador (através da tela) o qual irá gerar o arquivo xml contendo as informações modeladas. O processo pode ser observado na figura 10 a seguir.

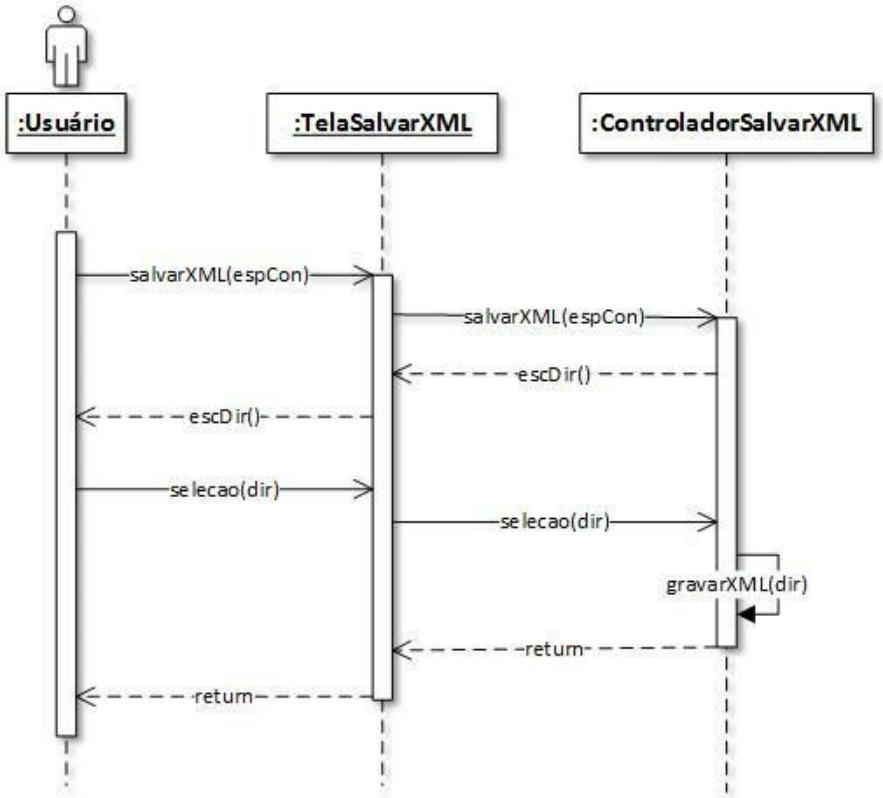

Figura 10. Operação salvar XML do projeto.

4.3.6 Carregar Arquivo XML:

O usuário solicita o carregamento de um arquivo XML a "TelaCarregarXML". A tela passa a solicitação ao controlador “ControladorCarregarXML”. Neste momento o controlado passa uma solicitação à tela, para que o usuário escolha o diretório onde está localizado o arquivo que será carregado. O usuário escolhe o arquivo, o diretório é passado da tela para o controlador, que então carrega a informação contida no XML e apresenta o resultado à tela onde será exibida. O processo descrito está representado na figura 11.

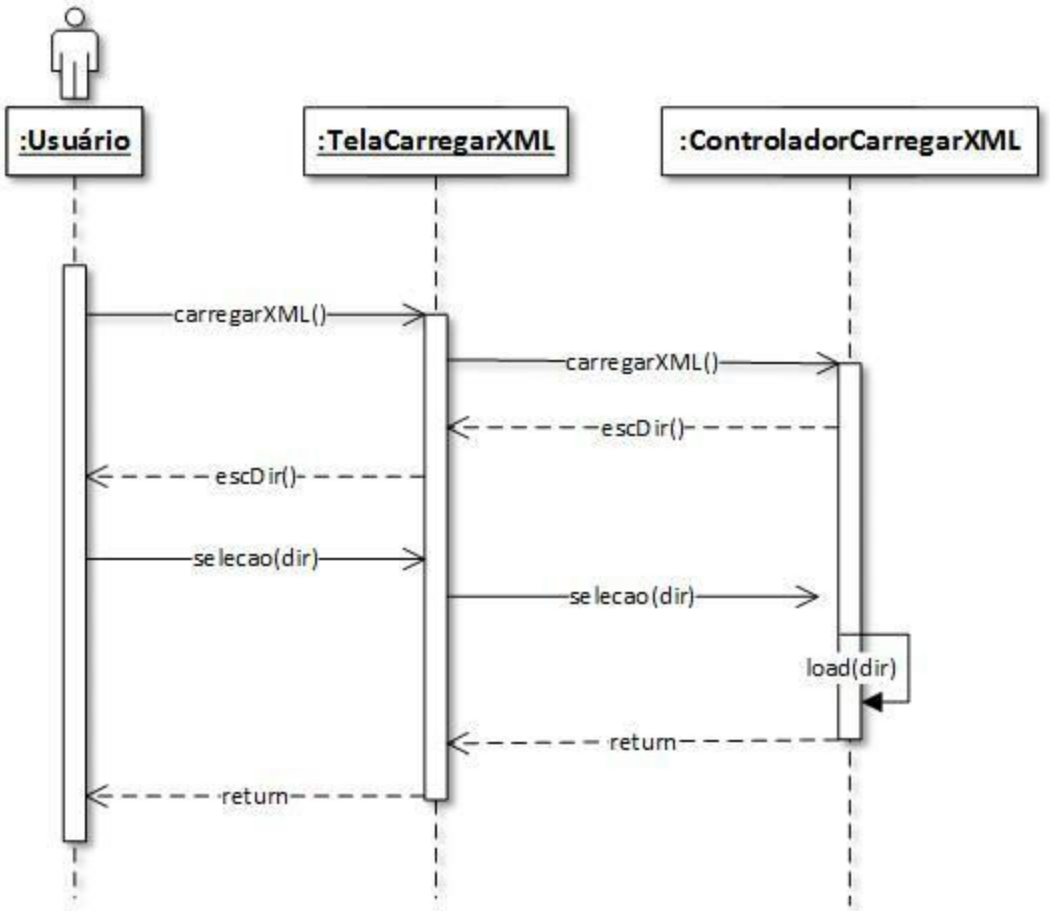

Figura 11. Operação carregar XML do projeto

4.4 Projeto e Implementação

4.4.1 Arquitetura

A partir do conjunto de diagramas de sequência da seção anterior, identificamos para cada caso de uso, dois elementos de análise: sua tela e os respectivos controladores. Não foi levantada nenhuma classe de entidade (a não ser o usuário), que precise ser salva de acordo com os casos de uso descritos, logo, o sistemas por enquanto não necessitará de elementos para armazenamento. As classes de análise foram então mapeadas para as classe de projeto de acordo com a figura 12. Nota-se que as camadas de interface Négocio-Dados e Dados estão fazias pelo fato anteriormente citado. Além dos controladores, podemos observar a existencia de um subsistema representando a ferramenta FDR 3. Como a ferramenta LFD-MPI não

implementa a verificação de deadlock, devemos ter uma interface de comunicação com esse sistema externo para podermos consumir seu serviço;

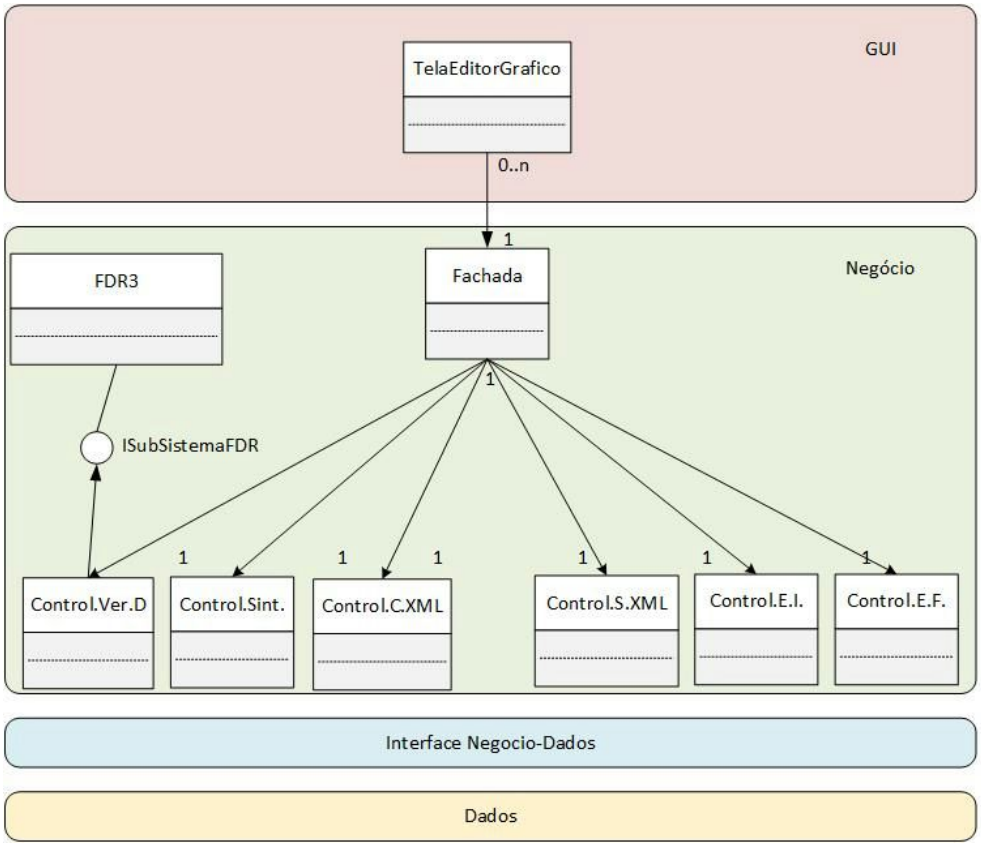

Figura 12. Arquitetura da Ferramenta.

4.4.2 Decisões de Projeto

4.4.2.1 Protótipo:

A figura 13 ilustra a prototipação da ferramenta LFD-MPI. Nela podemos identificar os principais elementos da ferramenta, que foram mapeadas da fase de analise para a fase de projeto. Por ser um editor gráfico, as decisões de projeto transformaram as telas levantadas durante a fase de análise em uma única tela, onde os casos de uso estão agrupados em barras de funcionalidades, caixas de texto e

elementos gráficos. A seguir serão descritos os principais pontos da ferramenta, e sua ligação com os casos de uso levantados na seção 4.2.

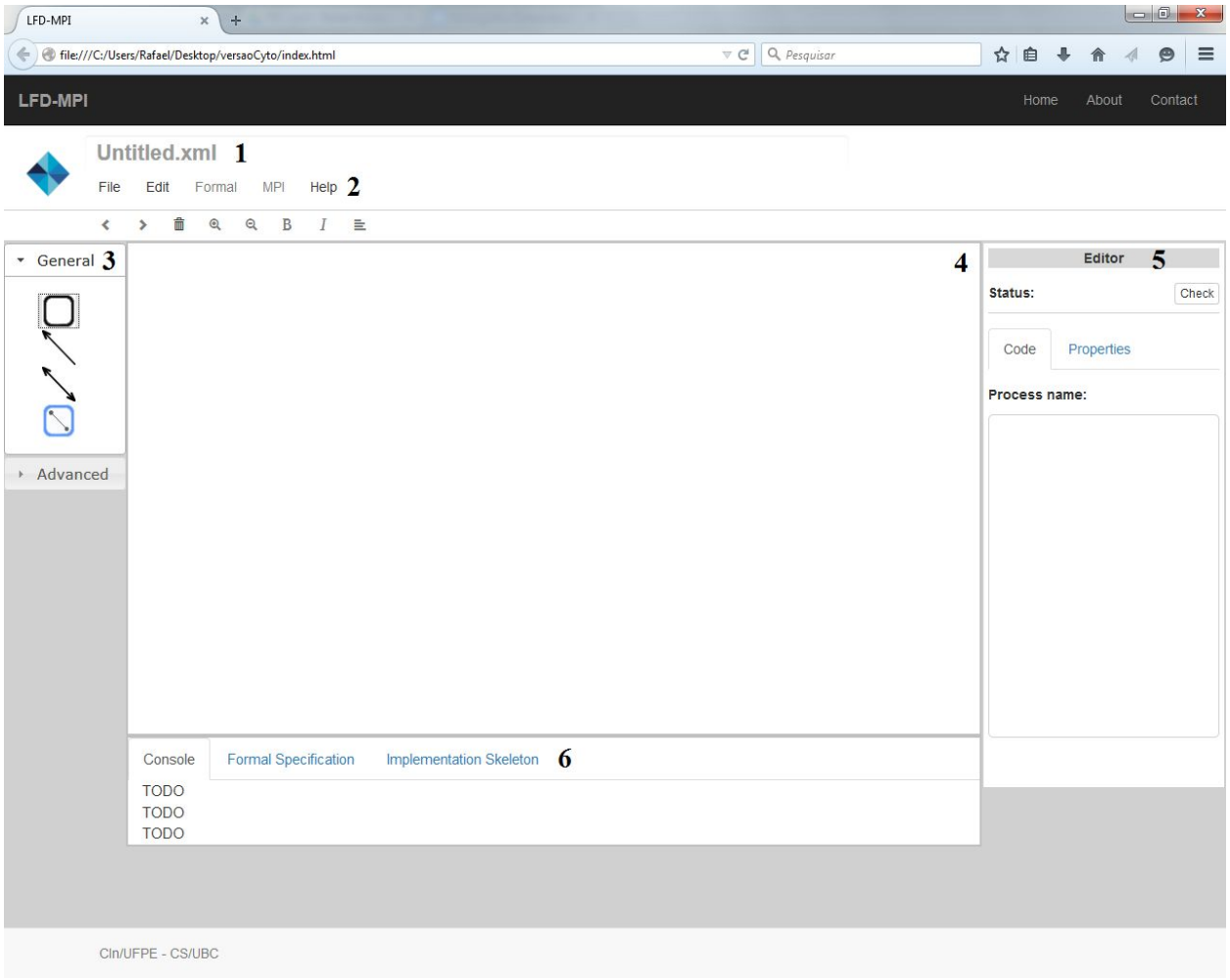

Figura 13. Protótipo da Ferramenta LFD-MPI.

1 - Nome:

Este campo irá apresentar o nome do arquivo XML contendo a modelagem feita pelo usuário. Este campo será atualizado automaticamente no carregamento de um arquivo, mas também possibilitará que o usuário defina um novo nome para o seu projeto.

2 - Barra de Funcionalidades:

A barra de funcionalidades é dividida da seguinte maneira:

- **File**: Um botão dropdown com opções New, permitindo que o usuário comece um novo projeto; Open, correspondendo ao caso de uso carregar

o arquivo XML “UC006”; Save, correspondendo ao caso de uso salvar o arquivo XML “UC005”.

- **Edit**: Um botão dropdown opções de edição, entre elas: Undo, Redo, Copy, Paste, Delete, etc. Essas opções também são consideradas funcionalidades futuras a ferramenta, logo a implementação, pode ou não contê-las.
- **Formal:** Este botão dropdown irá conter 2 opções: a primeira é o Generate permitindo que o usuário gere a especificação formal do seu programa satisfazendo o caso de uso “UC002”. A segunda opção é o Verify, onde a especificação formal será checada contra a presença de deadlock utilizando a ferramenta FDR, e assim satisfazendo o caso de uso “UC003”.
- **MPI:** Este botão permite que o usuário gere o esqueleto de implementação (em C) do seu programa utilizando as normas de MPI. Este item corresponde ao caso de uso “UC004”.

3 - Formas (Shapes):

Este campo irá conter os elementos (processos e conectores) utilizados na fase de design estrutural. O usuário poderá adicionar esses elementos ao canvas (item 4), para criar a representação visual do seu programa e assim facilitar o processo de desenvolvimento como um todo.

4 - Canvas:

O canvas é a àrea responsável por exibir os elementos adicionados ao projeto durante a fase de design estrutural. Assim como outras ferramentas de modelagem gráfica, este item tem como objetivo tornar o desenvolvimento do projeto mais legível, e consequentemente, mais fácil de detectar erros de modelagem.

5 - Editor:

O editor é o elemento do sistema responsável tanto pelo design de comunicação assim como a sua verificação sintática. Este item irá conter uma aba “Code” onde o usuário poderá definir as primitivas de comunicação para cada um dos elementos de processos adicionados (pelo item 3), dando assim um significado semântico ao seu design estrutural (definindo o design de comunicação). O editor ainda terá um campo responsável pela verificação sintática das primitivas utilizadas em cada processo satisfazendo o caso de uso “UC001” (Obs: a verificação semântica ainda é complexa e será considerada como uma funcionalidade futura). Por fim, o editor também possuirá uma aba chamada “Properties” onde o usuário poderá definir alguns atributos (cor, nome, etc.) aos elementos adicionados ao canvas.

6 - Saída dos resutados:

Esta região do projeto possui 3 abas: Console, Formal Specification e Implementation skeleton. Cada uma dessas regiões é responsável pela apresentação de uma informação especifica. O console irá apresentar informações de execução do programa, por exemplo, se a verificação de deadlock falhou ou teve sucesso, assim como informações sobre a verificação sintática do design de comunicação de um processo. As abas Formal Specification e Implementation Skeleton, como os nomes sugerem, irão apresentar o output das operações de geração das especificação formal e esqueleto de implementação.

4.4.2.2 Implementação das Funcionalidades:

Os próximos tópicos apresentam as ideias adotadas durante a implementação das funcionalidades da ferramenta LFD-MPI.

Elementos Gráficos:

Seguindo o fluxo do processo de desenvolvimento apresentado no tópico 3.1, o primeiro passo é definir um notação gráfica que permita que o usuário consiga expressar a modelagem estrutural de sua aplicação MPI. Para representação dos processos MPI assim como suas interações nesta primeira fase do projeto, foram definidos os seguintes elementos de acordo com a figura 14:

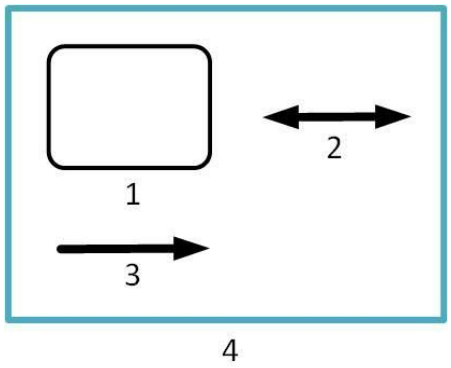

Figura 14. Elementos da modelagem estrutural.

1 - Processo: O retângulo preto com vértices arredondados, foi a notação escolhida para representação de um processo MPI na modelagem estrutural com a ferramenta.

Na ferramenta também será possível estilizar os processos com cores e nomenclaturas.
2 - Conector Duplo: A seta bidirecional representa a troca de informações entre dois processos.
3 - Conector Simples: A seta unidirecional foi utilizada para representar a troca de informações entre processos em apenas um sentido.
4 - Agrupamento: O retângulo azul indica o agrupamento de elementos.

Estrutura do arquivo XML:

Precisamos criar tags especiais para representar os elementos da seção anterior e que ao mesmo tempo permitam que o usuário consiga entender a informação contida dentro do arquivo. Para isso, foram definidas as seguintes tags:

- <ldf-specification>: Essa tag delimita o escopo da especificação de uma aplicação criada utilizando a ferramenta LFD-MPI.
- <lfd-component-process>: As tags de componentes adicionáveis ao canvas irão começar com o prefixo lfd-component, o próximo termo indicando o tipo de componente. Neste caso temos um processo.
- <lfd-component-conector>: Esta tag representa os conectores simples e duplo. Em versões futuras da ferramenta, esta tag irá conter informações sobre os processos envolvidos na conexão, porém ,na versão atual esta informação ainda não estará presente. Conectores simples, são identificados pelo ID “dsA#”(Division Simple Arrow), e os conectores duplos pelo ID “ddA” (Division Double Arrow).
- <lfd-component-group>: Tag que representa um agrupamento.
- <lfd-mpi>: Tag indicando o grupo de primitivas utilizadas em um processo.
- <lfd-primitive>: Tag que delimita uma primitiva.
- <lfd-pL>: Tag indicando os parâmetros (outros processos) utilizados na primitiva.
- <lfd-name>: Tag que indica o nome de uma primitiva.
- <lfd-x>: Posição horizontal de um elemento no canvas.
- <lfd-y>: Posição vertical de um elemento no canvas.
- <lfd-color>: Cor de um elemento expressa em RGB ou RGBA.
- <lfd-angle>: Ângulo em coordenadas transform (HTML) de um conector.

Uma aplicação onde temos dois processos “p1” e “p2”, representado na figura 15, em que p1 manda informações para p2, p2 as recebe e então responde p1, teria o formato apresentado na figura 16.

Figura 15. Aplicação com dois processos comunicantes.

```
<?xml version="1.0"?>
- <lfd-specification>
  - <lfd-component-process id="p2">
      <lfd-x>435px</lfd-x>
      <lfd-y>174px</lfd-y>
      <lfd-color>204, 255, 255</lfd-color>
    - <lfd-mpi>
      - <lfd-primitive>
          <lfd-name>MPI_Recv</lfd-name>
          <lfd-pl>p1</lfd-pl>
        </lfd-primitive>
      - <lfd-primitive>
          <lfd-name>MPI_Send</lfd-name>
          <lfd-pl>p1</lfd-pl>
        </lfd-primitive>
      </lfd-mpi>
    </lfd-component-process>
  - <lfd-component-process id="p1">
      <lfd-x>204px</lfd-x>
      <lfd-y>175px</lfd-y>
      <lfd-color>255, 153, 102</lfd-color>
    - <lfd-mpi>
      - <lfd-primitive>
          <lfd-name>MPI_Send</lfd-name>
          <lfd-pl>p2</lfd-pl>
        </lfd-primitive>
      - <lfd-primitive>
          <lfd-name>MPI_Recv</lfd-name>
          <lfd-pl>p2</lfd-pl>
        </lfd-primitive>
      </lfd-mpi>
    </lfd-component-process>
  - <lfd-component-connector id="ddA0">
      <lfd-x>279px</lfd-x>
      <lfd-y>187px</lfd-y>
      <lfd-angle/>
    </lfd-component-connector>
</lfd-specification>
```

Figura 16. Arquivo XML da aplicação.

Adaptador Formal:

"Assertions", são indicadores de quais propriedades o usuário gostaria de verificar em um programa CSPM. Na ferramenta FDR existe uma grande quantidade de "assertion" disponibilizadas para uso, inclusive, as que sinalizam a ferramenta, que o usuário deseja verificar a existência de deadlock em programa. FDR possui uma API java, logo é possível que o desenvolvedor crie um classe Java (importanto os arquivos ".jar" necessários), para fazer a verificação das propriedades formais. A figura 17 mostra o trecho de código onde é criada uma nova sessão FDR, em seguida é selecionado e carregado um arquivo com a extensão ".csp" nesta sessão. Então são executadas uma a uma as "assertions" definidas e seu resultado impresso. O código presente no adaptador formal é semelhante ao descrito na figura abaixo.

```
{
    try {
        Session session = new Session();
        session.loadFile("      .csp");
        for (Assertion assertion : session.assertions()) {
            assertion.execute(null);
            System.out.println(assertion.toString()+" "+
                (assertion.passed() ? "Passed" : "Failed"));
        }
    }
    catch (InputFileError error) {
        System.out.println(error);
    }
    catch (FileLoadError error) {
        System.out.println(error);
    }

    fdr.libraryExit();
}
```

Figura 17. Código do adaptador Formal.

Mapeadores C2F e C2I:

Como as regras de mapeamento ainda estão em processo de definição (pois são a parte mais complexa da ferramenta), até o momento da apresentação deste trabalho, serão prototipados alguns exemplos que demonstrem o funcionamento do sistema.

4.5 Tecnologias

A seguir serão apresentadas brevemente as tecnologias utilizadas durante a implementação da primeira versão web da ferramenta LFD-MPI. Elas foram divididas entre aquelas adotadas no frontend e as adotadas no backend do sistema.

4.5.1 Frontend

4.5.1.1 Bootstrap 3

Bootstrap[2] é um conjunto de ferramentas free e open-source [17] que permite a criação de websites e aplicações web [18]. Bootstrap 3 se propõe a facilitar e tornar mais rápido o desenvolvimento front-end, ele possui uma variedade de templates (baseados em HTML e CSS) de formulários, botões, estruturas de colunas, rodapés, cabeçalhos além de muitos outros componentes e extenções opcionais de javascript.

Bootstrap é modular e consiste essencialmente de um conjunto de definições de estilo LESS[3] (uma categoria de pré-processador de CSS) [19] as quais definem vários elementos da ferramenta e que podem ser modificadas para atender as necessidades específicas do programador.

Neste projeto foram utilizados elementos simples como botões e divisões estilizados para melhorar a aparência da ferramenta e consequentemente atender ao requisito não funcional “RNF003” mostrado no capítulo 4.

4.5.1.2 Jquery-UI

Assim como Bootstrap, Jquery-UI também é um framework porém sua função é facilitar o desenvolvimento em Javascript. Jquery-UI[4] fornece ao usuário um conjunto de interações de interface, efeitos de movimentação, widgets, temas, etc.

Neste projeto foram utilizados, listas em sanfona, abas, elementos draggable, para permitir que o usuário possa modelar e ter acesso às informações mais facilmente e satisfazer os requisitos não funcionais “RNF001” e “RNF002”.

[2] http://getbootstrap.com/
[3] http://lesscss.org/
[4] https://jqueryui.com/

4.5.1.3 Jquery-contextMenu

Jquery-contextMenu[5] é uma biblioteca javascript que permite o gerenciamento de elementos através de listas ordenadas. Ele é capaz de gerar automaticamente elementos DOM na forma listas com itens definidos pelo usuário seguindo as normas de implementação da sua documentação. Ele é disponível gratuitamente para download atráves do branch medialize/jQuerygitHub no gitHub.

Assim como as outras tecnologias citadas, contextMenu foi utilizado para permitir que o usuário compreenda melhor a ferramenta, uma vez que este elemento é bastante utilizado por diversos tipos de programas além dos de edição gráfica.

4.5.2 Backend

4.5.2.1 Java Server Faces (JSF)

Java Server Faces ou simplesmente JSF[6], é um framework MVC (model view controler) baseado em Java para desenvolvimento de aplicações web. JSF fornece vários controles de Interface baseados em HTML assim como os códigos que manipulam seus eventos. Ele pode ser usado para gerar graficos, formulários, e sua validação atráves da invocação de métodos da camada de negocios. JSF ainda pode ser usado em conjunto com diversas tecnologias de desenvolvimento frontend como primefaces, icefaces ou Bootstrap (escolhida para esse projeto).

Para atender o requisito não funcional “RNF004”, foram analisadas algumas alternativas de frameworks Java como Struts e Spring MVC, mas, baseado na curva de aprendizado de todas essas ferramentas e também considerando a complexidade da aplicação, foi definido a utilização de JSF 2.2.

Neste projeto foi criado um ManagedBean (POJO, ou Plain Old Java Object), chamado “AdaptadorFormal” (o adaptador responsável pela comunicação da ferramenta LFD-MPI com a ferramenta FDR) que realiza a chamada na camada de negocios, à ferramenta FDR para verificar se a especificação formal possui deadlock.

4.5.2.2 Webcomponents

Webcomponents são um grupo de 4 tecnologias (Custom Elements, HTML Templates, Shadow DOM e HTML Imports) que podem ser visto como um conjunto de widgets de interface de usuário (reutilizáveis) criados utilizando tecnologia open Web.

[5] https://github.com/medialize/jQuery-contextMenu
[6] https://javaserverfaces.java.net/

Por fazem parte do navegador, eles não precisam de bibliotecas externas, como jQuery ou Dojo para funcionarem.

Neste projeto foram utilizados elementos customizáveis para gerar as tags em html que representam a modelagem estrutural e de comunicação feita pelo usuário no arquivo XML.

4.5.2.3 Apache Tomcat

Apache Tomcat[7] é um servidor para aplicações web em Java. Ele foi desenvolvido pela Apache Software Foundation e atualmente é distribuído como software livre. Como foi utilizada a versão 2.2 do JSF neste trabalho, é preciso que a tecnologia de servidor tenha suporte à Java EE 6 ou superior. Existem outros servidores como Glassfish e o JBoss que também dão suporte à Java EE 6, mas foi decidido a utilização de Tomcat 8 para hospedagem da aplicação JSF.

[7] http://tomcat.apache.org/

Capítulo 5
Utilização da Ferramenta

A seguir, é descrito o processo de desenvolvimento, apresentado no capítulo 3, utilizando um protótipo da ferramenta LFD-MPI para ilustrar os passos desde a modelagem estrutural até a geração do esqueleto de implementação. Atualmente a ferramenta suporta as duas primeiras fases (design estrutural e design de comunicação), porém, os passos para a realização dos outros dois processos também serão descritos.

5.1 Tela Inicial

A primeira interação se dá no momento do acesso à ferramenta. Será gerada uma “modal view”, solicitando que o usuário dê um nome ao arquivo da modelagem que ele está prestes a desenvolver como mostrado na figura 18. Assumindo que chamamos o nosso arquivo de “Teste” e selecionamos o botão OK, seremos direcionado para a página principal da ferramenta. Todos os elementos mostrados na prototipação da ferramenta no capítulo 3 podem ser encontrados nesta tela.

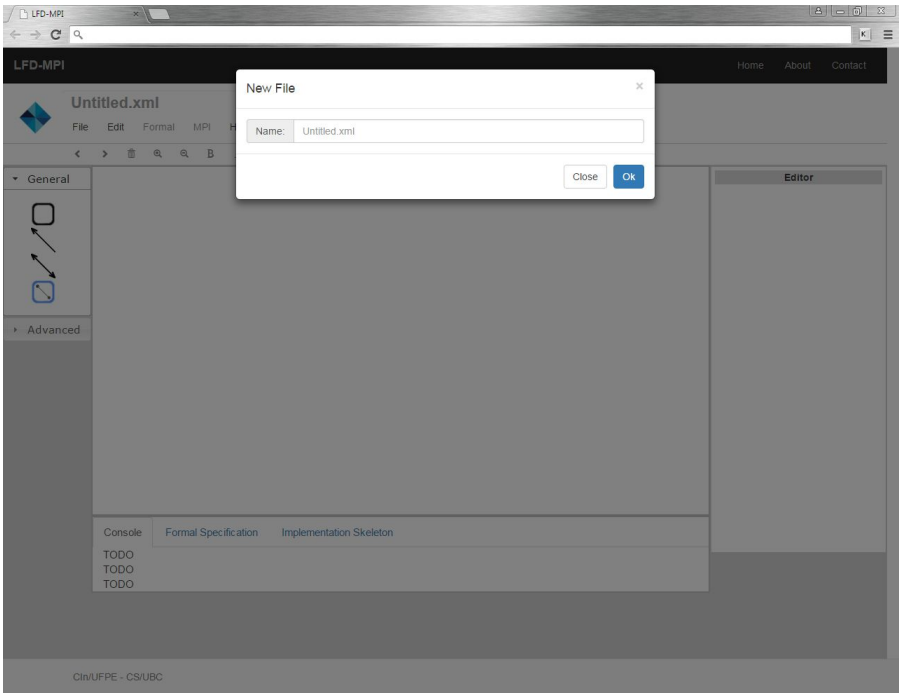

Figura 18. Tela inicial da ferramenta.

5.2 Design Estrutural

O primeiro passo do processo de desenvolvimento é o design estrutural do programa, nesta etapa, o usuário irá modelar graficamente a aplicação MPI. Para suportar este passo, a ferramenta LFD-MPI fornece 4 elementos gráficos: retângulo representando um processo, seta unidirecional representando uma comunicação assíncrona entre dois processos, uma seta bidirecional representando uma comunicação síncrona e por fim um retangulo azul que representa um agrupamento (esses elementos foram apresentados na seção de desenvolvimento da ferramenta).

No lado esquerdo da tela principal, teremos os elementos (processos, conectores, etc) que podem ser adicionado ao canvas (area central da ferramenta). Na figura 19, temos um design estrutural onde foram adicionados dois processos e um conector indicando a comunicação assíncrona entre eles.

Todos os elementos podem ser adicionados com um click, além disso, eles também são interativos, o usuário pode definir suas posições arrastando-os pelo canvas. Os conectores possuem um “handler” permitindo sua rotação para se adequar melhor à disposição dos outros elementos apresentados na tela.

Podemos alterar as propriedades de um processo clicando sobre ele e então alterando seus atributos na região do “Editor”. Tomando como exemplo o processo “p1” da figura anterior, teremos as abas Code e Properties. Na aba “Properties” é possível mudar informações de cor, transparência ou nome do processo; ja na aba “Code” teremos uma área de texto destinada ao design de comunicação da aplicação desenvolvida (a aba code será apresentada no próximo estágio de desenvolvimento).

As funcionalidades implementadas pelo design estrutural podem ser resumidas nas operações:

Adicionar elementos:
- Elementos podem ser adicionados ao canvas atraves de um clique.

Mover elementos:
- Qualquer elemento adicionado ao canvas pode ser arrastado para a posição desejada dentro do mesmo.

Alterar propriedade dos processos:
- Selecione o processo (já adicionado ao canvas) com um clique, em seguida, será apresentado no editor duas abas respectivamente a aba código, e a aba propriedades. Selecione a aba propriedades e altere as informações desejadas. Atualmente os conectores e os agrupamentos não possuem propriedades para serem alteradas, apenas os processos.

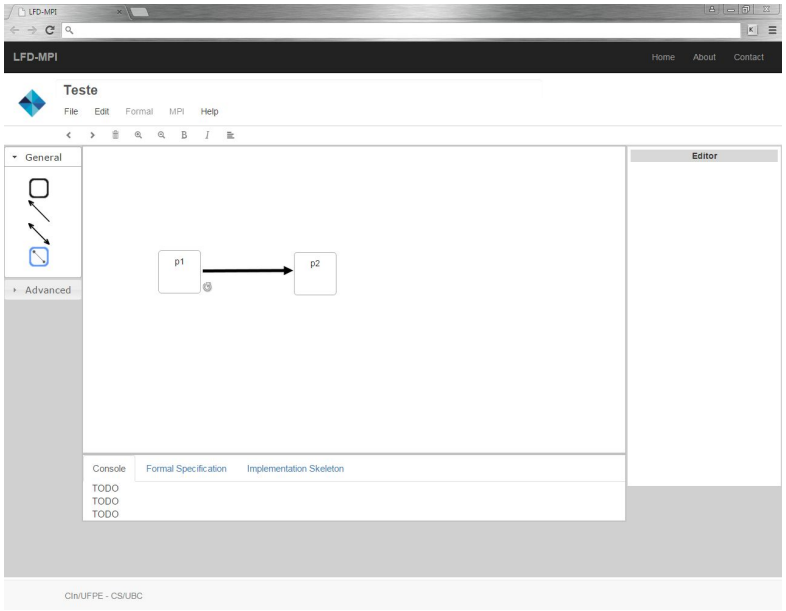

Figura 19. Modelagem estrutural com 2 processos e um conector.

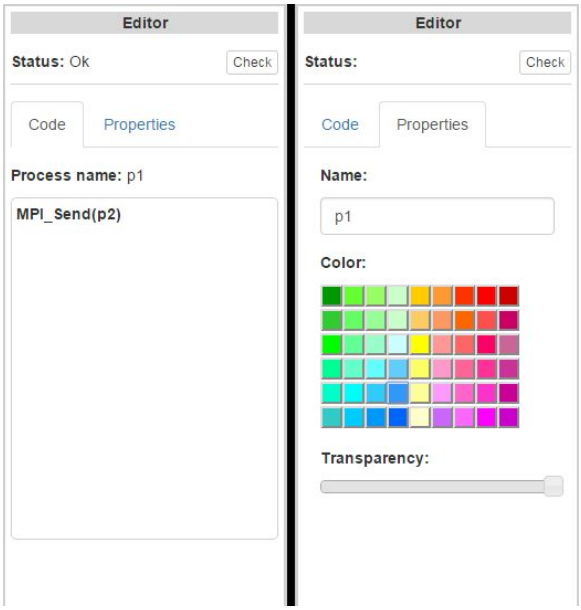

Figura 20. Informações dísponivel em cada aba do Editor.

Uma vez feita a modelagem do design estrutural, teremos a chamada de “especificação abstrata” do programa, que nada mais é do que a representação visual do programa MPI.

5.3 Design de comunicação

Continuando o processo de desenvolvimento, o segundo passo é fazer a modelagem de comunicação da aplicação, ou seja, atribuir um sentido à modelagem estrutural.

No passo anterior foi mencionado a existência de uma aba “Code” na região do editor quando clicamos em um processo (esta aba pode ser vista na figura 20 do lado esquerdo). Nessa aba existe uma área de texto destinada à programação do comportamento dos processos que formam o ambiente da aplicação (foram adicionados no passo da modelagem estrutural).

O comportamento dos processos é descrito utilizando-se as primitivas de MPI. Inicialmente a ferramenta dará suporte a utilização de sete primitivas que são: "MPI_Send", "MPI_Recv", "MPI_IRecv", "MPI_Wait", "MPI_Init", "MPI_Finalize", "MPI_Barrier"; essas primitivas são consideradas importantes pois praticamente toda aplicação MPI utilizam-as, dessa forma, iremos trabalhar com esse subconjunto de funções. Para cada processo será atribuido uma combinação dessas primitivas, separadas por uma quebra de linha, que irá descrever o comportamento do processo no ambiente. A figura 21 mostra um possível cenário de programação do comportamento de dois processos “p1” e “p2”. Nela temos o seguinte comportamento dos processos: “p1” manda uma mensagem para “p2” através da primitiva “MPI_Send(p2)”, “p2” recebe a mensagem de p1 através da primitiva “MPI_Recv(p1)”; em seguida “p2” responde a “p1” com o MPI_Send(p1), que por sua vez recebe a mensagem de “p2” com MPI_Recv(p2).

É importante notar que a ferramenta também dá suporte à verificação sintática do design de comunicação. Em cima da área de texto onde o usuário codifica o comportamento dos processos, existe um botão chamado “check”, este botão implementa o caso de uso “UC001". Quando o botão é acionado, o controlador responsável pela verificação sintática das primitivas irá capturar o texto digitado na área do codigo, então ele irá comparar cada uma das linhas (lembre-se que a separação entre uma instrução e outra é uma quebra de linha) com o conjunto das sete primitivas definidas no sistema. Se o código digitado não conter erros sintáticos, será exibida uma mensagem (ao lado do botão check, na área chamada “status”), com o indicador “Ok”, caso contrário, será exibido “Not-Ok”; um exemplo de verificação pode ser visto na figura 20.

Portanto, as funcionalidades da fase de design de comunicação são:

Implementar o comportamento dos processos:
- Cada processo pode ter seu comportamento modelado utilizando-se o conjunto de primitivas "MPI_Send", "MPI_Recv", "MPI_IRecv", "MPI_Wait", "MPI_Init", "MPI_Finalize", "MPI_Barrier"; associado com os parâmetros necessários para cada uma dessas funções.

Verificação Sintática:
- Para continuar o processo de desenvolvimento utilizando a ferramenta LFD-MPI, é necessário, antes de gerar uma especificação em CSP, realizar uma verificação sintática das primitivas utilizadas, dessa forma o mapeamento para CSP (proxima fase do desenvolvimento) ocorrerá sem problemas. Para realizar a verificação, o usuário deverá (após programar o comportamento dos processos utilizando as primitivas MPI) clicar no botão “Check” na aba código. Se o código estiver corretamente escrito, será retornado “Ok”, caso contrário, “Not-ok”.

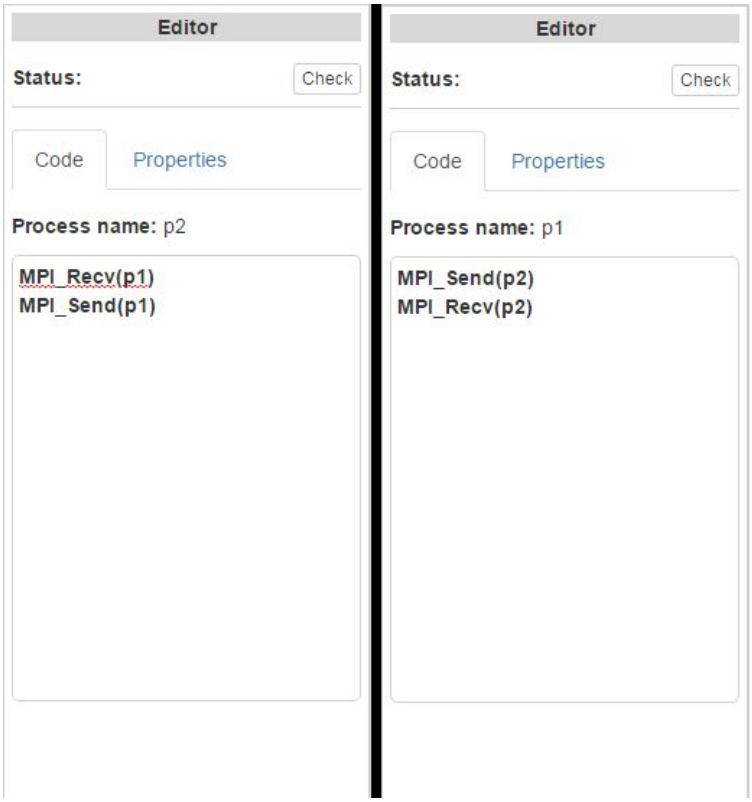

Figura 21. Design de comunicação de uma aplicação com dois processos.

Assim que o usuário tiver completado o design de cominucação e sua verificação sintática, ele terá a especificação concreta (artefato da segunda fase do pocesso de desenvolvimento) da aplicação.

5.4 Formalização

O próximo passo é a formalização da especificação concreta. Esta fase é composta por duas etapas: a geração do código CSP a partir da especificação concreta; e a verificação de deadlock (esta é a única propriedade formal que queremos verificar).

Feita a modelagem de comunicação dos processos, precisamos gerar um código CSP que será a fornecido como entrada para a ferramenta FDR 3 que realiza a verificação de propriedades formais (no nosso caso estamos apenas interessados em checar deadlock). Para gerar a código CSP o usuário deverá selecionar a opção Formal > Generate, então o mapeador C2F (concrete to formal) irá realizar as operações necessárias para traduzir a especificação concreta em código CSP. Esta operação pode ser observada na figura 22.

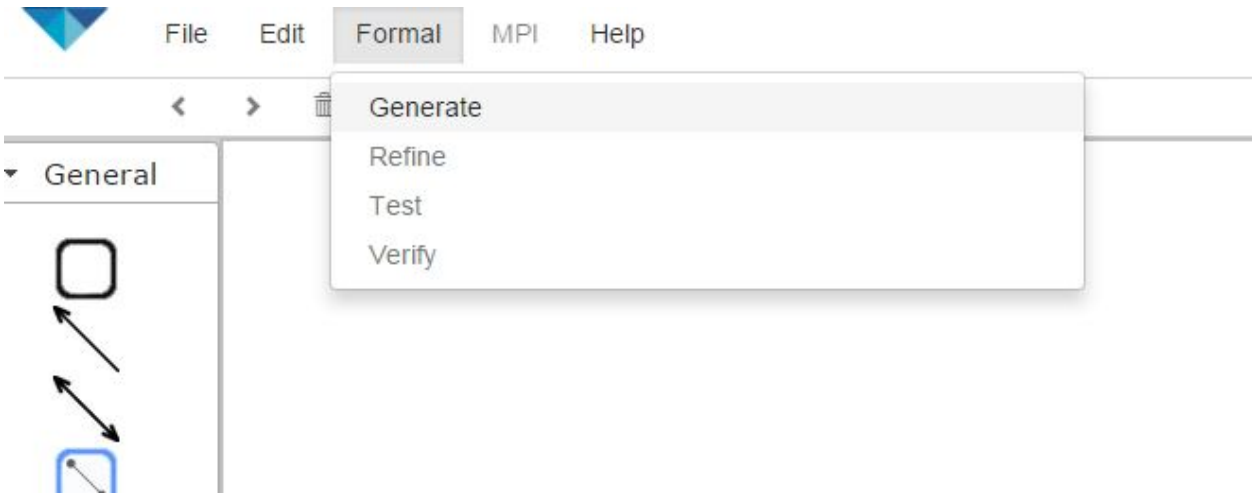

Figura 22. Gerar código CSP a partir da especificação concreta.

.

O resultado desta operação será impresso na aba “Formal Specification” abaixo do canvas (figura 23). Ressaltamos que as regras de mapeamento da notação utilizada para fazer o design de comunicação em processos CSP ainda estão em processo de desenvolvimento, logo, a versão atual da ferramenta tem o objetivo de apenas prototipar o funcionamento da versão final esperada.

Figura 23. Impressão do código CSP na aba Formal Specification.

.

Nesta etapa, poderiamos ter outros mapeadores para outras linguagens, por exemplo: poderiamos ter um mapeador para para ACP (Bergstra & Klop's *Algebra of Communicating Processes*), outro para CCS (Milner's *Calculus of Communicating Systems*), para isso, basta implementar diferentes módulos que poderia ser acessados por uma interface que, dependendo da linguagem que o usuário escolhesse, iria chamar funções desses módulos para converter a especificação concreta, na notação desejada.

Ainda na fase de formalização, nós precisamos checar se o nosso código CSP (que representa nossa modelagem MPI) possui deadlock, para isso, selecionamos a opção verify da aba Formal (figura 24).

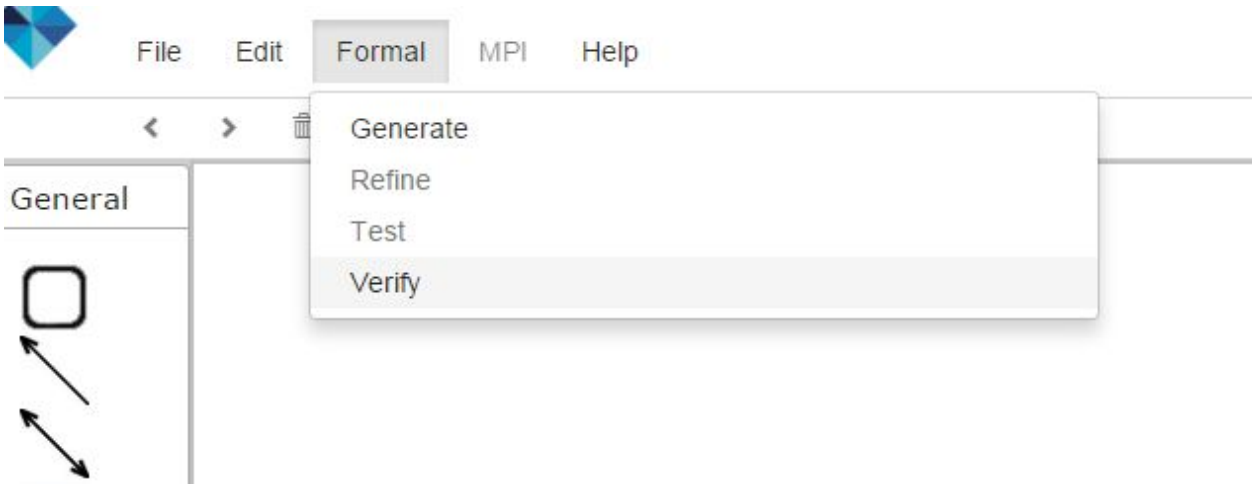

Figura 24 Verificação de Deadlock**.**

Se o resultado da verificação for positivo, ou seja, não existe deadlock, o usuário poderá passar para a próxima fase gerar o esqueleto de implementação, caso contrário, o usuário precisará voltar a fase de modelagem de comunicação e estudar melhor a solução para o seu problema, possivelmente também realizar alguma alteração na modelagem estrutural. As opções “Refine” e “Test” da aba “Formal” são operações futuras feitas no código CSP, porém elas ainda estão em processo de definição. Portanto a etapa da verificação funcional apresenta as seguintes funcionalidades:

Gerar código CSP:
- Gerar o código CSP que será testado na ferramenta FDR 3 para verificação de deadlock. O usuário deve selecionar a opção “Formal > Generate“ na barra de funcionalidades.

Verificar Deadlock:
- Alimentar a ferramenta FDR 3 com o código CSP gerado através da especificação concreta. Esta funcionalidade pode ser acessada através da opção “Formal > Verify” desde que a notação CSP já tenha sido formada.

5.5 Mapeamento de Implementação

Após a fase de formalização, o usuário poderá gerar o esqueleto de implementação selecionando a opção “MPI > Generate” (figura 25) na barra de funcionalidades. O mapeador C2I (concrete to implementation) irá operar sobre a especificação concreta criando então um esqueleto de implementação em C que será apresentado na aba Implementation Skeleton. Vale lembrar que esta operação ainda não é suportada pela ferramenta e que a aba Console é reservada para imprimir informações a cerca de qualquer operação realizada durante o processo de desenvolvimento.

Durante o processo de modelagem em qualquer momento o usuário poderá salvar o seu projeto escolhendo a opção File > Save. Esta operação irá gerar o XML contendo a modelagem estrutural e de comunicação feita até o momento. Caso o usuário queira carregar um arquivo de modelagem feita anteriormente, ele pode escolher a opção File > Open e selecionar o arquivo desejado.

Com isso, fechamos a apresentação da ferramenta. Algumas funcionalidades ainda precisam ser refinadas e também adicionadas, porém, os processos básicos foram estes descritos.

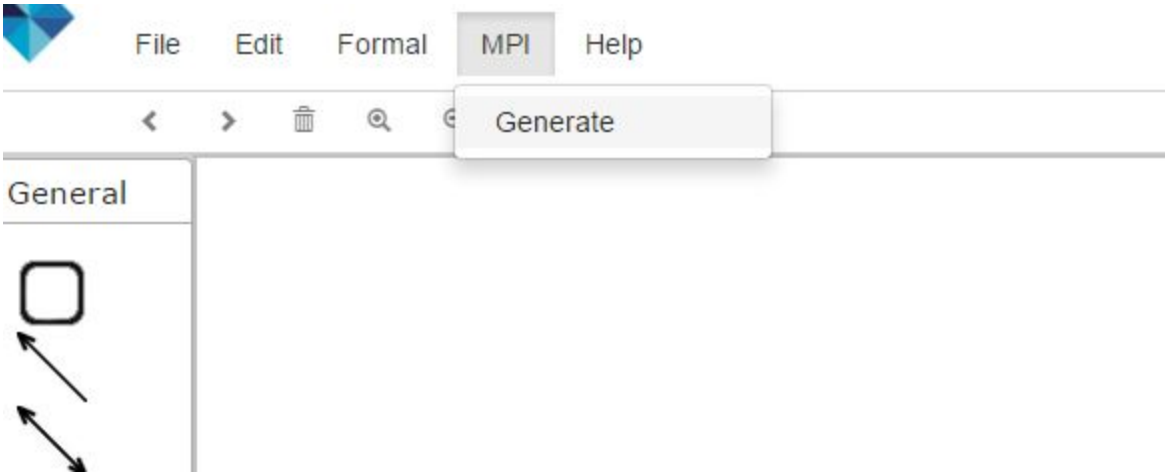

Figura 25 Função gerar esqueleto de implementação MPI**.**

Capítulo 6

Conclusão

Constatamos com [8, 9] que várias áreas do conhecimento tem utilizado aplicações baseadas em MPI, porém como apresentado em [10,11], muitas soluções tem sido desenvolvidas para auxiliar a difícil tarefa de implementar um programa utilizando esta descrição. Erros relacionados ao uso incorreto de seus recursos, erros de aplicação que causam impasse (deadlock), são bastante comuns, além da falta de informação que reforcem o seu uso correto contribui para o desenvolvimento de aplicações falhas. Desta forma, a existência de uma ferramenta que ajude o desenvolvedor neste processo e que verifique propriedades formais do código desde os estágios iniciais do desenvolvimento, torna-se uma solução bastante atrativa e competitiva em relação às descritas em [10,11].

LFD-MPI é uma ferramenta com o propósito de sanar esses problemas associados ao desenvolvimento com a especificação MPI. Ela fornece um editor gráfico que permite a visualização geral da aplicação através de elementos simples como conectores e processos (representados por setas e figuras geométricas), facilitando o entendimento do programa e consequentemente a correção de erros. Associado com ferramentas externas LFD-MPI ainda é capaz de verificar uma importante propriedade formal dos programas desenvolvidos (deadlock), apesar dessas ferramentas também serem capazes de verificar outras propriedades, nos concentramos apenas na questão do impasse. Ela ainda é capaz de verificar a sintaxe das primitivas MPI utilizadas no projeto, informando ao usuário se ele está utilizando alguma delas de maneira incorreta.

A análise semântica também é uma funcionalidade muito importante almejada pela ferramenta LFD-MPI, por exemplo, verificar se o desenvolvedor está utilizando algum parâmetro que não existe. Esta função ainda está em fase de desenvolvimento, porém é um item indispensável para atingir seu proprósito (facilitar o desenvolvimento com MPI). Atualmente a ferramenta suporta as duas primeiras etapas do processo de desenvolvimento, além dos casos de uso “UC005” e “UC006”, salvar e carregar um arquivo XML contendo um projeto criado dentro da ferramenta. As outras etapas ainda estão em fase de definição e desenvolvimento.

Analisando a proposta inicial deste trabalho de graduação tínhamos a seguinte estrutura: Introdução, Descrição da Ferramenta, Desenvolvimento, Tecnologias, e Conclusão. Esta estrutura foi modificada da seguinte maneira: foi adicionado um capítulo para a introdução do leitor aos conceitos iniciais (capítulo 2), e os capítulos de

desenvolvimento e tecnologias foram agrupados na descrição da ferramenta (capítulo 4); e por último, foi adicionado um capítulo de utilização da ferramenta (capítulo 5).

Um dos principais desafios na elaboração deste trabalho, foi o levantamento dos requisitos da ferramenta (e seu entendimento como um todo) associado com a necessidade de aprendizado das tecnologias envolvidas no seu desenvolvimento. Foram necessárias muitas horas de estudos de diferentes tecnologias, para definir quais seriam utilizadas entre as candidatas. Além disso, muitas horas de estudo para realizar a implementação dos módulos do projeto, o que acabou prejudicando o desenvolvimento total da ferramenta. Até o momento, não foi possível definir as regras de mapeamento dos componentes C2F e C2I apresentados no capítulo 3, dessa forma, até o momento da apresentação deste trabalho, será realizada a prototipação de exemplos de mapeamento, para ilustrar o funcionamento do programa.

Por ser uma ferramenta que facilita o trabalho do usuário, a complexidade do seu desenvolvimento aumenta consideravelmente. Ainda existem diversas funcionalidades que precisam ser implementadas para tornar a ferramenta mais amigável e competitiva no mercado, portanto, o desenvolvimento desta ferramenta ainda precisa ser estendido por alguns anos.

Um requisito importante levantado, foi a hospedagem deste trabalho no domínio do Centro de Informática da Universidade Federal de Pernambuco, porém, como mencionado no tópico referente ao Apache Tomcat (capítulo 4), é preciso que o ambiente onde a aplicação está hospedada tenha suporte à Java EE 6 ou superior. Atualmente é possível rodar a aplicação a partir do Eclipse, com hospedagem local, mas ainda é preciso encontrar uma solução para a tornar a ferramenta pública, e na página do “Group on Foundations and Applications of Distributed Systems”[8].

[8] https://sites.google.com/a/cin.ufpe.br/gfads/

Referências Bibliográficas

[1] R. Nelson, W. Alan; “A Lightweight Approach to Formally Develop and Verify MPI Applications”.

[2] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, J. Dongarra, “MPI: The Complete Reference”, MIT Press, Cambridge, MA, USA, 1996.

[3] A. Bernardo, “Parallel Computation”, In: Computer Physics Communications, 56 25-42 (1989).

[4] Baeten, J.C.M. "A Brief History of Process Algebra". Theoretical Computer Science 335 (2005) 131 – 146.

[5] B. Blaise; “Introduction to Parallel Computing”. Lawrence Livermore National Laboratory. Disponível em: <https://computing.llnl.gov/tutorials /parallel_comp/>. Acessado em: 07 de Maio de 2015.

[6] J. Peter, F. Walter; ”Highly Parallel Computation”, Science 250, 1217-1222, (1990).

[7] “Message Passing Paradigm”. OVPIT AFS CELL Indiana Univesity. Disponível em :<http://beige.ucs.indiana.edu/I590/node13.html>. Acessado em : 10 de maio de 2015.

[8] S. Vakkalanka; “Efficient Dynamic Verification Algorithms for MPI Applications”. Ph.D. thesis, Salt Lake City, UT, USA (2010), aAI3413092.

[9] A. Emmanuel, C. Camille, H. Thomas, L. Julien, P. Sylvain, R. Ala,C. Franck, D. Jack; “QCG-OMPI: MPI applications on grids”. Future Generation Computer Systems. 27, 357-369, (2011).

[10] S. Vakkalanka, A. Vo, G. Gopalakrishnan, K. Robert; “Precise dynamic analysis for slack elasticity: Adding buffering without adding bugs”. Em: Proceedings of the 17th European MPI Users’ Group Meeting Conference on Recent Advances in the Message Passing Interface. pp. 152–159. EuroMPI’10, Springer-Verlag, Berlin, Heidelberg (2010).

[11] T. Hilbrich, J. Protze, M. Schulz, Supinski, B.R.d., Muller, M.S.: MPI runtime error detection with MUST: Advances in deadlock detection. Scientific Programming 21(3), 109–121 (2013).

[12] B. Blaise; “Message Passing Interface”. Lawrence Livermore National Laboratory. Disponível em: <https://computing.llnl.gov/tutorials/mpi/>. Acessado em 10 de Maio de 2015.

[13] Roscoe, A. W. “The Theory and Practice of Concurrency”. Prentice Hall. ISBN 0-13-674409-5. (1997).

[14] K. Gerald, S. Ian. “Requirements Engineering: Processes and Techniques”, John Wiley & Sons, 1998.

[15] S. Ian. “Engenharia de Software”. 6ª Edição, Makron Books, 2003.

[16] M. James. “An Information Systems Manifesto”. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[17] "open source." *Encyclopaedia Britannica. Encyclopaedia Britannica Online Academic Edition*. Encyclopædia Britannica Inc., 2015. Web. 14 Jun. 2015.

[18] C. Alex; "What is a web application”. Disponível em: <http://www.jguru.com/faq/view.jsp?EID=12932 8>. Acessado em: 03 de Julho de 2015.

[19] F. Thoriq; “LESS CSS - Begginer’s Guide”. Disponível em:<http://www.hongkiat.com/blog/less- basic/>. Acessado em 01 de Maio de 2015.

[20] L. Anghel, “Mastering Java Server Faces 2.2: Master the art of implementing user interfaces with JSF 2.2”. PacktPub, Junho de 2014