UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

BRAULIO ADRIANO DE MELLO

Co-Simulação Distribuída de Sistemas Heterogêneos

Tese apresentada como requisito parcial
para a obtenção do grau de Doutor em
Ciência da Computação

Prof. Dr. Flávio Rech Wagner
Orientador

Porto Alegre, maio de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Mello, Braulio Adriano de

Co-Simulação Distribuída de Sistemas Heterogêneos / Bráulio Adriano de Mello. – Porto Alegre: PPGC da UFRGS, 2005.

145f.: il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2005. Orientador: Flávio Rech Wagner.

1.Simulação heterogênea distribuída. 2. Backbone de simulação. 3. Cooperação. 4.DCB. I. Wagner, Flávio Rech. II. Título.

AGRADECIMENTOS

Ao Prof. Dr. Flávio Rech Wagner. A postura criteriosa e séria no acompanhamento das atividades, a organização e a proximidade perante as orientações foram tão importantes quanto as demonstrações de incentivo e segurança no período em que o auto-controle e a determinação foram mais exigidos. São nos momentos em que quaisquer das nossas perspectivas parecem perder o significado é que mais precisamos delas. Meus agradecimentos aos que me ajudaram a manter as perspectivas (professores, colegas, familiares, amigos,...). Muitas vezes vi, num simples cumprimento, um gesto sutil de aprovação e apoio pelo esforço que estava sendo feito para continuar com as atividades normais (mesmo que num ritmo menor). Sou bastante grato por isso.

Aos professores que participaram das bancas de avaliação (proposta de tese e/ou tese), prof. Fabiano, prof. Bernardo, prof. Carlos Eduardo, prof. Luigi, prof. Susin, pelas contribuições importantes na evolução e fechamento da tese. Ao Ernesto, Josué, Uilian, Daniel B., Daniel W., Lúcio, Brião, amigos que, no desenvolvimento de atividades acadêmicas junto ao Instituto, contribuíram direta ou indiretamente com esta tese. As reuniões, presenciais e virtuais, foram importantes para a concretização de idéias e definição de metas, não apenas técnicas, mas também de vida. Aos amigos que reencontrei e encontrei no Instituto, Márcio K., César Z., Edgard... Ao esforço conjunto da UFRGS, PPGC, corpo docente, funcionários, coordenações, promovendo o desenvolvimento acadêmico, da ciência, do país. À universidade de origem, a URI, pelas condições que permitiram o ingresso e conclusão do curso. Aos amigos aqui da URI com quem pude contar em muitas situações.

Aos meus pais, Bento e Maria Helena, e meu irmão, Paulo. A família na essência do equilíbrio. À família da Mônica (minha esposa), a quem considero também minha. Aos familiares em quem encontrei apoio para assumir desafios, ao Mauricio, meu primo, pela serenidade em horas de aflição, ao Lourenço. Ao Carlinhos (em memória), meu tio, de quem tive verdadeiros exemplos de vida, e a sua família.

À Mônica, minha esposa, sempre presente. À minha filha, Ana Carolina, com quem estou aprendendo, a cada dia, a entender a simplicidade e o significado da vida.

SUMÁRIO

LISTA DE SIGLAS

ACT	*Abstract Control Types*
AMG	*Architecture Management Group*
API	*Aplication Programming Interface*
CAD	*Computer Aided Design*
CGI	*Common Gateway Interface*
CORBA	*Common Object Request Broker Architecture*
CORSA	*Component-ORiented Simulation Architecture*
CSP	*Communicating Sequential Processes*
CT	*Continuous Time*
DCOM	*Distributed Component Object Model*
DCS	*Dynamic Component Substituition*
DDE	*Distributed Discrete Event*
DE	*Discrete-Event*
DEVS	*Discrete Event System Specification*
DIS	*Distributed Interactive Simulation*
DMSO	*Defense Modeling and Simulation Office*
DoD	*US Department of Defense*
DT	*Discrete Time*
EDCB	Embaixador do DCB
EF	Embaixador do Federado
EJB	*Enterprise JavaBeans*
FLI	*Foreign Language Interface*
FOM	*Federation Object Model*
FSM	*Finite-State Machines*
GVT	*Global Virtual Time*
HILS	*Hardware-In-the-Loop Simulation*
HLA	*High Level Architecture*
HTML	*Hyper Text Markup Language*

IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Intellectual Property / Internet Protocol*
IPANEMA	*Integration Plataform for Networked Mechatronic Applications*
JFP	*JavaCAD Foundation Packages*
LAN	*Local Area Network*
LCC	*Local Causality Constraint*
MCI	*Multilanguage Distributed Co-Simulation ToolInterface*
MDA	*Model Driven Architecture*
OMG	*Object Management Group*
OMT	*Object Model Template*
PADS	*Paralell and Distributed Simulati*on
PL	*Logical Process*
PN	*Process Networks*
PVS	*Prototype Verification System*
RMI	*Remote Method Invocation*
RTI	*Run-Time Infrastructure*
RTL	*Register Transfer Level*
RTOS	*Real Time Operating System*
S^3E^2S	*Specification, Simulation, and Synthesis of Embedded Electronic Systems*
SDF	*Synchronous Dataflow*
SIMNET	*SIMulator NETworking*
SoC	*System-on-Chip*
SOM	*Simulation Object Model*
SR	*Synchronous/Reactive*
SSL-IF	*System Specification Language-Intermediate Form*
TCP	*Transport Control Protocol*
TL	*Transfer Level*
UML	*Unified Modeling Language*
VHDL	*Very High Speed Integrated Circuits Hardware Description Language*
VLSI	*Very Large Scale Integration*
WAN	*Wide Area Network*
WESE	*Web-based Environment Systems Engineering*
XML	*eXtensible Markup Language*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Na simulação heterogênea de um sistema eletrônico complexo, um mesmo modelo pode ser composto por partes distintas em relação às tecnologias ou linguagens utilizadas na sua descrição, níveis de abstração, ou pela combinação de partes de software e de hardware (escopo da co-simulação). No uso de modelos heterogêneos, a construção de uma ponte eficaz entre diferentes simuladores, em conjunto com a solução de problemas tais como sincronização e tradução de dados, são alguns dos principais desafios. No contexto do projeto de sistemas embarcados, a validação desses sistemas via co-simulação está sujeita a estes desafios na medida em que um mesmo modelo de representação precisa suportar a cooperação consistente entre partes de hardware e de software.

Estes problemas tornam-se mais significativos quando abordados em ambientes distribuídos, o que aumenta a complexidade dos mecanismos que gerenciam os ítens necessários à correta cooperação entre partes diferentes. Contudo, embora existam abordagens e ferramentas voltadas para o tratamento de modelos heterogêneos, inclusive em ambientes distribuídos, ainda persiste uma gama de limitações causadas pela distribuição e heterogeneidade de simuladores. Por exemplo, restrições quanto à variedade de tecnologias (ou linguagens) utilizadas na descrição das partes de um modelo, flexibilidade para o reuso de partes existentes, ou em tarefas de gerenciamento de sincronização/dados/interface/distribuição. Além disso, em geral, nas soluções existentes para simulação heterogênea, alterações são necessárias sobre as partes do modelo, limitando a preservação de sua integridade. Esta é uma característica indesejável, por exemplo, no reuso de componentes IP (*Intellectual Property*).

Neste contexto, esta tese apresenta o DCB (*Distributed Co-simulation Backbone*), cujo propósito geral é o suporte à execução distribuída dos modelos heterogêneos. Para isso, são observados de modo integrado quatro fatores básicos: a distribuição física; a independência dos componentes (partes); o encapsulamento das estratégias de gerenciamento de tempo, de dados e de comunicação; e a sincronização híbrida. Em geral, as soluções existentes valorizam um fator em detrimento dos demais, dependendo dos propósitos envolvidos e sua variação em relação ao grau de especificidade (soluções proprietárias ou restritas a um escopo de aplicações).

O Tangram, também discutido nesta tese em termos de requisitos, é uma proposta de ambiente para projeto de modelos heterogêneos distribuídos. No contexto da especificação do DCB, esta proposta tem como objetivo geral agregar num mesmo ambiente funcionalidades de apoio para a busca e catalogação de componentes, seguidas do suporte à construção e à execução distribuída de modelos heterogêneos via DCB. À luz dos princípios de generalidade e flexibilidade da arquitetura do DCB, o Tangram visa permitir que o projetista reduza seu envolvimento com detalhes relacionados ao provimento de condições necessárias à cooperação entre componentes heterogêneos.

No escopo desta tese, ênfase foi dada à co-simulação de sistemas embarcados, ênfase esta observada também na construção do protótipo do Tangram/DCB, e nos estudos de caso. Contudo, a estrutura do DCB é apropriada para qualquer domínio onde a simulação possa ser utilizada como instrumento de validação, entre outros propósitos.

Palavras-chave: simulação heterogênea distribuída, *backbone* de simulação, cooperação

Distributed Co-simulation of Heterogeneous Systems

ABSTRACT

In the simulation of complex embedded systems, the same model could be composed by heterogeneous parts, considering various languages or technologies used in its description, abstraction levels, or combinations of hardware and software components (co-simulation). In the application of heterogeneous models, the construction of a bridge between different simulators and the solution of problems like synchronization and data translation are some of the main challenges. The validation of embedded systems by co-simulation cannot avoid these challenges, because a representation model needs to support a consistent cooperation between hardware and software parts.

These problems are even harder to handle in distributed environments, where mechanisms for controlling the communication between different parts become more complex. Although there are approaches and tools oriented towards heterogeneous models, and particularly considering geographically distributed environments, they still present various limitations mainly due to the distribution and heterogeneity of simulators. As an example, there are limitations imposed on the possible technologies (or languages) that are used for the description of model parts, on the reuse flexibility, or on tasks for managing synchronization/data/interface/distribution. Moreover, the existing solutions reduce the integrity of the components (and thus reduce reusability) because they generally require modifications in the component models. This is an undesirable feature, for example, for IP (Intellectual Property) reuse.

This thesis presents DCB (Distributed Co-simulation Backbone), an infra-structure for supporting the distributed execution of heterogeneous models. DCB offers four main features in an integrated way: physical distribution of components; independence of components; strategies for encapsulation of time, data, and communication management tasks; and hybrid synchronization. In general, existing solutions only partially consider these requirements, neglecting some of them according to the needs of particular application domains.

This thesis also discusses requirements for the development of Tangram, a design environment for distributed heterogeneous models. This proposal is based on the DCB architecture. Its goal is to gather, in a single environment, capabilities to search and catalog components and to support the construction and distributed execution of heterogeneous models. Tangram follows the DCB characteristics regarding generality and flexibility, in order to reduce the designer's effort in providing conditions for the cooperation between heterogeneous components.

This thesis emphasizes embedded systems as the main domain of co-simulation while it presents Tangram/DCB and experimental results. However, the DCB

architecture could be used in any other domain where distributed simulation could be useful as a validation mechanism.

Keywords: distributed heterogeneous simulation, simulation backbone, cooperation

1 INTRODUÇÃO

A evolução dos sistemas computacionais vem agregando maiores exigências aos recursos utilizados na sua representação. São exemplos destes recursos as linguagens e/ou técnicas de especificação. Esta evolução tem contribuído com o incremento da complexidade dos sistemas de tal forma que o uso de apenas uma linguagem, ou nível de abstração (nível de detalhamento), tem se mostrado insuficiente. Observa-se este fato principalmente em etapas intermediárias do processo de projeto de sistemas, onde o uso de estratégias de validação baseadas em descrições multi-linguagem pode contribuir com a detecção precoce de erros, permitindo que eles sejam solucionados de modo mais fácil e ágil.

A combinação de componentes heterogêneos e a distribuição estão entre as principais causas do aumento de complexidade do processo de projeto e a conseqüente redução de produtividade no projeto de sistemas [GER2002]. Sistemas que combinam componentes diferentes que precisam cooperar são chamados de heterogêneos [HUB98]. Os componentes de um sistema podem ser diferenciados em termos de tecnologias ou de linguagens utilizadas na sua descrição, em termos de representação em níveis de abstração diferentes, ou ainda como software ou hardware (escopo da co-simulação). Além disso, os componentes podem estar geograficamente distribuídos, e/ou também sob restrições de acesso/uso (propriedade intelectual), o que aumenta a complexidade com que projetistas de sistemas precisam se defrontar. Restrições de acesso são principalmente adotadas nos casos onde há solicitações de uso por terceiros. Mesmo quando todos os módulos de um sistema são desenvolvidos por grupos pertencentes a um mesmo projeto ou empresa, ainda é expressiva a probabilidade de ocorrência de erros na cooperação entre estes módulos.

Por isso, nas fases que envolvem o projeto/implementação de sistemas heterogêneos, é natural que os componentes sejam desenvolvidos e validados em separado e, por vezes, através de processos também diferenciados. Nestes casos, os primeiros testes de funcionamento de um sistema inteiro, considerando a comunicação entre os componentes desenvolvidos, geralmente ocorrem apenas nas etapas finais do projeto (implementação). O retardo na validação de sistemas heterogêneos pode postergar a descoberta de problemas que, se detectados em fases mais iniciais, poderiam ser resolvidos com menor esforço/custo.

Deste modo, é bastante desejável que a busca de erros (validação) na cooperação em sistemas heterogêneos [SCH2003] seja realizada também em etapas intermediárias do processo de desenvolvimento desses sistemas. Contudo, tal validação tende a alcançar um custo considerável devido à complexidade inerente de sistemas com componentes heterogêneos e/ou distribuídos, podendo inclusive superar os benefícios de um trabalho de validação. O custo da validação aumenta proporcionalmente à

indisponibilidade de estratégias/técnicas adequadas para sua execução (principalmente de estratégias automatizadas).

Para a validação da interação entre os componentes de um sistema heterogêneo ainda nas fases de projeto, a simulação [LAW91] apresenta-se como um mecanismo bastante eficaz. Contudo, os ambientes ou ferramentas de simulação utilizados precisam suportar tal heterogeneidade. Deste modo, visando tanto aspectos de validação das interações, bem como a avaliação do comportamento dos componentes individualmente, a abordagem denominada de simulação heterogênea tem ocupado crescente espaço em pesquisas [DAL99].

A simulação heterogênea pode ser utilizada em sistemas com propósitos variados. Por exemplo, no projeto de sistemas que combinam componentes de hardware e software (co-simulação de sistemas embarcados [LEE2001b]), ou na construção de ambientes de treinamento onde existe a interação entre componentes de software e participantes humanos. Contudo, é importante observar que diferentes domínios, onde a simulação heterogênea pode ser utilizada, possuem requisitos e restrições que variam de acordo com as propriedades e exigências desses domínios. Por isso, as metodologias e ferramentas de simulação precisam estar preparadas para tratar adequadamente os requisitos de cada domínio em particular. Tais requisitos podem ser mensurados em termos de desempenho, volume de informações, necessidades de sincronização, distribuição, etc.

O suporte à execução cooperativa de simuladores diferentes é o princípio fundamental da co-simulação [HES99]. Um simulador pode ser um sistema computacional, um recurso de hardware, uma parte real do sistema, ou mesmo um participante físico/vivo. Na composição de um modelo de co-simulação, cada simulador assume a responsabilidade pela representação de uma das partes que compõem um sistema em estudo. Abordagens composicionais [CHO98a] [CHO98b] estimulam o reuso de componentes.

As características de heterogeneidade e distribuição de um sistema precisam ser incorporadas pelo modelo de simulação que o represente. Assim, os simuladores podem ser executados em um único nodo, ou em múltiplos nodos. Quando executados em múltiplos nodos, eles podem fazer parte de um ambiente distribuído local (LAN - *Local Area Network*), ou de uma WAN (*Wide Area Network*) [HIN98]. Em função deste segundo caso, recursos de comunicação via Web precisam ser utilizados, aumentando o número de variáveis que precisam ser controladas. Entre elas, a maioria dos estudos em simulação distribuída aborda problemas de sincronização do relógio de simulação [FUJ92] mantido pelos simuladores.

Neste cenário, cada simulador pode enviar e receber dados através de uma interface de co-simulação que precisa gerenciar a comunicação, a sincronização e a tradução de dados para formatos reconhecidos por simuladores destino. A literatura, em geral, apresenta a construção de uma ponte eficaz entre simuladores heterogêneos como um dos maiores desafios em co-simulação [KIM96].

Ao mesmo tempo em que são propostas novas e melhores contribuições para a validação de sistemas heterogêneos através da construção e execução de modelos heterogêneos, aumentam também os níveis de complexidade. Esta tendência é principalmente observada em modelos cujos componentes (ou sub-modelos) são distribuídos em nodos distintos de uma rede de computadores, seja local ou geograficamente distribuída. Neste contexto, afora o aumento de complexidade no gerenciamento de simulações, a distribuição tende a agregar maior flexibilidade para o

desenvolvimento de projetos, por exemplo promovendo o reuso de software e o compartilhamento de recursos de desenvolvimento (software, hardware e recursos humanos) de forma mais expressiva.

Em contrapartida, torna-se mais difícil obter resultados consistentes, em tempo hábil, sem o auxílio de mecanismos ou instrumentos de apoio. Esta dificuldade deriva da existência de múltiplas arquiteturas, técnicas, linguagens, etc, que passam a interagir num ambiente de projeto distribuído sob diferentes situações e condições. Por isso, investimentos no desenvolvimento de estratégias, tais como no campo da co-simulação de HW e SW, têm despertado crescente interesse. O gerenciamento de questões de propriedade intelectual e o desenvolvimento descentralizado de projetos também são fatores que têm motivado o desenvolvimento de pesquisas nesse tema, principalmente em ambientes geograficamente distribuídos.

As pesquisas no campo da co-simulação têm contribuído para a consolidação de uma teoria básica consistente envolvendo conceitos e técnicas sobre comunicação, requisitos de sincronização em diferentes domínios de aplicação, interfaces, desempenho, modelagem, entre outros [HUB98] [CHO98a] [YOO98] [HIN97a] [YOO97] [SPO2001] [KIM96] [HOF2001]. Mais do que isso, novos ambientes e ferramentas de projeto de sistemas que oferecem recursos de co-simulação têm sido propostos. Observando as linhas gerais de ambientes/ferramentas existentes, eles podem ser diferenciados pelas características ou estratégias que utilizam para a solução de problemas e pelos domínios de aplicação para os quais são desenvolvidos. Tais observações, em geral, são relatadas em estudos de casos em co-simulação [NIC2002].

Contudo, em geral, os ambientes e ferramentas existentes não incorporam características de generalidade com o objetivo de tratar problemas inesperados, ou não previstos, em um determinado domínio. Deste modo, soluções proprietárias ou baseadas em um dado conjunto de restrições podem impor um excessivo fardo em termos de reprojeto de ferramentas. Mais do que isso, o uso explícito de soluções não proprietárias e distribuídas, em conjunto, não tem sido visto como uma questão fundamental nas abordagens existentes.

Exemplos de tais condições restritivas podem ser encontrados em abordagens existentes, como por exemplo: uso de técnicas particulares para especificação e síntese de modelos, como em IPCHINOOK [CHO99]; uso de funções ou estruturas pré-definidas, como em JavaCAD [DAL99] e Pia *sockets* [HIN98]; limitações no número ou tipos de linguagens, como em arquiteturas/ferramentas multilinguagens [CHA2001] [HES99] [HES2001] (geralmente voltadas também para a síntese); e interfaces proprietárias de comunicação, como em algumas soluções baseadas em *backplanes* [SUN98]. Estes fatores têm motivado a busca por alternativas mais flexíveis, p.ex. SSA (*Standard Simulation Architecture*) [STE2003].

Outro aspecto que motiva estudos em simulação heterogênea é a reutilização de componentes já desenvolvidos em outros projetos, sejam próprios ou de terceiros. Nestes casos, se forem úteis num novo projeto, os componentes podem possuir interfaces inconsistentes uma vez que podem ter sido desenvolvidos em situações diferenciadas [WAG2004]. No contexto do reuso de componentes de propriedade intelectual (IP-*Intellectual Property* [PAT2002]) [DAL99] a simulação heterogênea distribuída contribui com benefícios tais como a simulação remota do componente IP, proteção [DAL2000], reuso independente da tecnologia de construção do componente e a redução do tempo de projeto [KEA2002] [REU99].

1.1 Objetivos

No escopo do estado da arte em simulação de sistemas heterogêneos, este trabalho de tese tem como objetivo geral propor mecanismos que ofereçam contribuições para facilitar e agilizar a construção e execução de modelos heterogêneos de simulação em ambiente distribuído [MEL2001a] [MEL2002] [SOU2003]. A estrutura de suporte à execução aqui proposta, denominada DCB (*Distributed Co-simulation Backbone*), visa agregar características de generalidade e flexibilidade, permitindo que diferentes sub-modelos de simulações existentes possam ser adicionados em um mesmo modelo de co-simulação, sem impor alterações sobre a estrutura interna destes sub-modelos (ou componentes).

As estratégias de preservação dos componentes previstas no DCB, por sua vez, visam não impor limites na variedade de tecnologias, linguagens, ou níveis de abstração de modelos existentes, para integrá-los mediante finalidades comuns. Nesta perspectiva, condições são criadas pelo DCB para que se possa manter o suporte à execução de modelos heterogêneos independentemente dos componentes, evitando que a tecnologia de construção de componentes interfira na estrutura de suporte à execução.

A idéia fundamental da arquitetura do DCB é remover as restrições que são impostas sobre simuladores independentes, considerando aspectos de interface, cooperação, e sincronização. Em comparação com HLA (*High Level Architecture*) [DAH97], arquitetura a partir da qual o DCB foi inspirado, o DCB adiciona uma característica essencial: enquanto em HLA os *federados* (termo utilizado para designar os componentes da simulação distribuída) precisam estar preparados para requisitar serviços ao RTI-*Real Time Infrastructure* (a estrutura de suporte à execução de modelos HLA) de modo explícito, no DCB esta tarefa é retirada dos federados e assumida por mecanismos intermediários previstos na sua arquitetura, denominados embaixadores.

Esta tática busca atingir o objetivo de oferecer condições reais para que um federado seja adicionado a um modelo de co-simulação sem a necessidade de adaptações ou alterações na sua estrutura interna. Assim, preservando a integridade de cada simulador, o DCB pode mais facilmente tratar situações inesperadas (p.ex. incorporação de um novo componente implementado em tecnologia não prevista no ambiente em estudo), sem comprometimento do restante do modelo.

À luz da arquitetura do DCB, este trabalho de tese discute requisitos para uma proposta de ambiente de auxílio na construção de modelos de co-simulação. Este ambiente tem como metas principais a busca e catalogação de componentes, a construção de modelos através da composição de componentes em um ambiente visual, e a configuração do modelo resultante visando sua execução com suporte do DCB.

Também é visto como objetivo de igual importância permitir que um modelo de co-simulação possa ser executado em ambiente distribuído. Tendo seus componentes cooperantes alocados a diferentes nodos, o DCB deve prover consistência em termos de sincronização (tempo de simulação) e gerenciamento de dados (troca de valores entre componentes). A simulação distribuída de modelos homogêneos tem sido bastante explorada na bibliografia, o que ainda não ocorre para modelos de co-simulação. A necessidade de tradução de dados para o formato reconhecido por componentes-destino tende a exigir mais esforço dos mecanismos que gerenciam a execução distribuída de co-simulações.

1.2 Contribuições

As contribuições desta tese, no que tange ao suporte à execução de modelos heterogêneos em ambiente distribuído, podem ser decompostas em quatro itens gerais: a distribuição física dos componentes; a preservação de detalhes internos dos componentes; o encapsulamento das estratégias de gerenciamento de tempo, de dados e de comunicação; e a sincronização híbrida.

Estudos em co-simulação são principalmente voltados para execução centralizada. Além disso, geralmente o projetista tem acesso aos detalhes internos dos componentes que precisam cooperar, fato este que oferece alternativas ao projetista para contornar a heterogeneidade. Esta tarefa pode ser baseada no esforço humano ou na ajuda de ferramentas de apoio que, em geral, possuem aplicação específica (proprietárias). Ferramentas para HILS (*Hardware in the loop simulation*) [ZAN2000] [STO98], por exemplo, são de aplicação específica. Como exemplo, a plataforma IPANEMA (*Integration Plataform for Networked Mechatronic Applications*) [HON95] segue o conceito de HILS e é direcionada para a execução centralizada de testes em sistemas que agregam partes mecânicas e elétricas (sistemas mecatrônicos), além de sistemas de controle.

No entanto, na execução distribuída, a heterogeneidade de modelos tende a exigir instrumentos adicionais para manter a consistência, dispensáveis em modelos homogêneos, principalmente quando é necessário manter a 'originalidade' dos componentes (quando observados aspectos de propriedade intelectual). Um exemplo é a necessidade de tradução de dados para formatos reconhecidos pelo componente-destino. Neste contexto, o DCB promove a execução de *federações* (termo usado para designar um modelo de simulação distribuído formado por componentes que podem ser heterogêneos) cujos federados podem estar fisicamente distribuídos.

Quanto à independência de componentes, a arquitetura do DCB tem como prioridade a preservação da integridade do componente. Para isso, o primeiro princípio é de que o federado visualiza um único módulo externo com quem se comunica, tratando apenas aspectos de interface. Ou seja, o componente permanece alheio às operações de gerenciamento da federação que ocorrem além do módulo de interface do DCB, denominado *gateway*. Estas operações ocorrem em módulos internos do DCB, que, por sua vez, desconhecem o componente, pois visualizam apenas as primitivas do *gateway* para atualização de valores de atributos.

Um outro fator positivo, resultante desta característica do DCB, é a redução do tempo necessário para a construção de um modelo heterogêneo. Ou seja, a eliminação da necessidade de alterações em detalhes internos do código dos componentes, substituída pela política de configuração e geração de *gateways*, reduz o esforço de modelagem.

O encapsulamento de estratégias de gerenciamento mantido pelo DCB está diretamente ligado aos propósitos de independência dos componentes. A completa separação entre detalhes internos dos componentes e as políticas de gerenciamento de dados e de tempo, entre outras, necessárias à execução de um modelo heterogêneo e distribuído de simulação, promove tal independência. Esta característica contribui grandemente para o reuso de componentes.

Em relação à sincronização, o DCB permite que componentes síncronos e componentes que não modelam explicitamente a passagem do tempo (denominados '*notime*') possam participar simultaneamente de um mesmo modelo. Embora não implementado no protótipo construído no escopo desta tese, o DCB prevê também a cooperação de componentes que evoluem no tempo de forma assíncrona. Em geral, pesquisas em simulação distribuída concentram esforços no provimento de cooperação entre componentes síncronos e assíncronos apenas e essencialmente para simulações homogêneas.

Em relação ao suporte para a construção de modelos heterogêneos, este trabalho discute um conjunto de requisitos com vistas à especificação e construção de um ambiente de modelagem para sistemas heterogêneos. Definido a partir da arquitetura do DCB, tais requisitos prevêem o atendimento de funcionalidades de apoio para a busca e catalogação de federados, de construção e de execução de modelos heterogêneos. Baseado nos princípios de generalidade e flexibilidade da arquitetura do DCB, este conjunto de requisitos visa a construção de um ambiente (denominado 'Tangram') cujo objetivo geral é a redução do envolvimento do projetista com detalhes relacionados ao provimento de condições necessárias à cooperação entre federados heterogêneos.

No suporte à modelagem de sistemas baseados em componentes como previsto nos requisitos para a construção do Tangram, é verdade que funcionalidades semelhantes são oferecidas em ambientes já existentes. Um exemplo é o SIMOO [COP97], também desenvolvido no PPGC-UFRGS. No entanto, mecanismos de suporte à composição de componentes heterogêneos e remotos agregam complexidade a esta atividade, o que justifica o aparecimento de situações que exigem mais funcionalidades no suporte à modelagem, abrangendo desde a busca e catalogação de componentes heterogêneos até sua execução distribuída.

As propriedades do DCB/Tangram incorporam princípios importantes no trato de ambientes heterogêneos, uma vez que tratam de aspectos relevantes relacionados à comunicação, sincronização e tarefas de tradução de dados entre simuladores [BIS97] [HUB98].

1.3 Estrutura do texto

O Capítulo 2 apresenta uma introdução sobre os conceitos fundamentais de simulação distribuída e co-simulação. Além da discussão conceitual, este capítulo apresenta alguns ambientes existentes de propósitos similares ao Tangram/DCB, contribuição desta tese. O Capítulo 3 apresenta a arquitetura do DCB, argumentando sobre os propósitos e as estratégias do DCB visto como um mecanismo de suporte à execução de modelos de co-simulação em ambientes distribuídos.

O Capítulo 4 tem como objetivo a apresentação de requisitos para a construção de um ambiente de modelagem denominado Tangram. São discutidos requisitos cujos propósitos giram em torno da busca, catalogação e composição de componentes para a construção e execução de modelos heterogêneos.

O protótipo do DCB, com a implementação de algumas funcionalidades do Tangram, construído neste trabalho de tese, é apresentado no Capítulo 5.

O Capítulo 6 apresenta o estudo de caso desenvolvido, discutindo resultados alcançados/esperados com o uso do Tangram/DCB. Neste capítulo são abordados quatro

experimentos realizados com modelos que representam o comportamento de um dispositivo de monitoração (utilizado em meios de locomoção) denominado SistemaGPSAlerta [LIS2002]. Em cada um dos experimentos o modelo do SistemaGPSAlerta sofre alterações que adicionam heterogeneidade e maior detalhamento.

O Capítulo 7, finalmente, apresenta as conclusões e perspectivas para trabalhos futuros.

2 CO-SIMULAÇÃO DISTRIBUÍDA NA VALIDAÇÃO DE SISTEMAS HETEROGÊNEOS

Este capítulo tem como propósito apresentar uma revisão sobre os conceitos gerais a respeito da simulação no contexto da distribuição e da heterogeneidade de modelos. Os princípios de simulação distribuída, seguidos de aspectos de projeto para construção de modelos de simulação distribuída, são discutidos. Também é propósito deste capítulo uma apresentação de trabalhos referenciados nesta tese e de uma análise comparativa entre estes.

2.1 Introdução

O propósito básico da simulação é permitir a execução de atividades/estudos de sistemas com o uso de modelos de representação. Um modelo incorpora propriedades do sistema que representa, possuindo então a capacidade de 'imitar' seu comportamento. Quanto maior o número de detalhes do sistema real incorporados ao modelo, maior sua fidelidade, contudo maior o volume de dados que precisam ser tratados.

A bibliografia cita benefícios variados com a adoção de técnicas de simulação. Entre eles [LAW91]:

- A atividade de construção de modelos para sistemas existentes contribui para o entendimento e a identificação de problemas;
- Através de modelos é possível alcançar projeções futuras, com bons níveis de precisão e considerando o uso de probabilidade, a partir de um comportamento conhecido;
- No projeto de sistemas novos, a construção de um modelo de simulação permite observar o comportamento do sistema antes mesmo de sua implementação;
- A simulação pode ser vista como indispensável na construção/reestruturação de sistemas onde testes ou experimentos em situações reais são muito caros ou inviáveis;
- Seja na construção ou reestruturação de sistemas, a simulação permite que sejam aplicadas atividades de validação em diferentes níveis do processo de projeto de sistemas.

No contexto do projeto de sistemas computacionais, a simulação está principalmente vinculada a propósitos de validação de sistemas desde fases precoces do desenvolvimento. Transcendendo a simulação centralizada, onde um modelo é inteiramente executado em um único recurso de processamento, modelos podem ser executados de modo particionado. Ou seja, um modelo é dividido em partes cooperantes que podem ser executadas de forma paralela ou distribuída [FUJ95] [FER95] [FUJ2001].

A simulação paralela tem como objetivo maior o ganho de desempenho na execução de modelos. Para isso, em geral, esta linha se refere ao uso de plataformas de computação multiprocessadas. Já na simulação distribuída, desde que mantida uma taxa mínima suficiente de desempenho de acordo com os objetivos da simulação, as partes de um modelo podem ser executadas em diferentes sistemas computacionais autônomos interligados por redes de comunicação. A preocupação com realismo, distribuição, flexibilidade, cooperação, entre outros, é valorizada em detrimento da busca por desempenho. Tais computadores podem estar em um mesmo local, ou geograficamente distribuídos, o que também agrega benefícios tais como facilidades para o reuso, a convivência com questões de proteção de componentes, trabalho cooperativo envolvendo equipes fisicamente distribuídas, entre outros [FUJ99] [FUJ2000].

Considerando modelos particionados, pesquisas em simulação têm questionado a construção de modelos cujas partes são construídas com tecnologias distintas que, contudo, precisam cooperar. Aqui denominada de simulação heterogênea, esta linha da simulação aumenta a abrangência dos benefícios que a simulação pode oferecer. Além disso, na simulação, seja ou não distribuída, a complexidade de um modelo pode alcançar níveis elevados o suficiente para que o seu desenvolvimento não possa mais ser realizado com o uso de apenas um único recurso de desenvolvimento, por exemplo uma linguagem de descrição ou um único nível de abstração.

A simulação heterogênea é bastante explorada no escopo dos sistemas embarcados, onde a integração entre hardware e software (*codesign*) é inerente [CHO98a]. Este tipo de sistema geralmente é caracterizado por agregar de forma cooperativa e simultânea recursos de hardware e de software, ou partes com diferentes níveis de abstração, ou ainda múltiplas linguagens de desenvolvimento que sejam adequadas à representação de cada uma de suas diferentes partes [HUB98].

Citada na bibliografia como co-simulação de sistemas embarcados, esta linha de pesquisa tem como propósito geral oferecer condições para que estes sistemas possam ser validados por inteiro em diferentes fases do processo de projeto (níveis de abstração), antes da sua implementação [LEE2000]. Justifica-se esta tendência na construção de sistemas heterogêneos pois, geralmente, as partes distintas são desenvolvidas e validadas separadamente.

A validação de um sistema inteiro significa levar em consideração também os aspectos de comunicação e os efeitos da interação entre componentes. Seguindo os conceitos gerais de verificação e validação de sistemas, quanto mais precoce a identificação de problemas, mais facilmente, rapidamente e a menores custos, eles podem ser resolvidos. A complexidade desta tarefa é proporcional à heterogeneidade das partes que interagem [BER2002].

Em resumo, o princípio básico da co-simulação é a execução cooperativa de simuladores diferentes, onde cada simulador é responsável pela execução da

representação de uma parte do sistema em desenvolvimento. Nestes estudos, têm sido enfatizadas questões sobre o provimento da comunicação e interface entre partes heterogêneas [HUB98]. Os simuladores podem ser executados em uma mesma máquina ou em múltiplas máquinas. Quando múltiplas máquinas são utilizadas, elas podem estar localizadas em uma rede local ou geograficamente distribuída [HIN98].

2.2 Co-simulação de sistemas embarcados heterogêneos

No contexto de sistemas embarcados, um dos maiores problemas encontrados no uso de modelos de co-simulação está relacionado à construção das interfaces entre hardware e software. Sua construção torna-se mais complexa na medida em que os níveis de abstração utilizados incorporam mais detalhes no decorrer do processo de *co-design*. Deste modo, um sistema em desenvolvimento pode ser inicialmente simulado utilizando um modelo que representa (imita) o comportamento do sistema em alto nível de abstração. Em seguida, ocorrem etapas de refinamento (decorrentes do processo de *co-design*) onde são alcançados níveis com uma cobertura de detalhes próxima do sistema real e, finalmente, a própria implementação do sistema.

Simulações em alto nível de abstração normalmente são executadas nas fases iniciais do projeto. Elas têm como objetivo geral validar a funcionalidade dos componentes e suas interações. Isto tende a promover a redução do tempo de projeto e da ocorrência de erros [CES2002]. Nessas simulações, a interface entre hardware e software precisa ser modelada em alto nível de abstração utilizando alternativas para comunicação entre componentes (por exemplo, primitivas *send* e *receive*). Esse tipo de simulação é geralmente composto por um modelo de representação do comportamento do hardware e pelo protótipo simplificado do software. Esta abordagem, que não leva em conta aspectos de sincronização, contribui com a redução do tempo de validação [PAD2000], mas é pouco útil para tarefas de avaliação de desempenho e verificação final do sistema.

Já em níveis mais baixos de abstração, a interface entre componentes pode ser modelada em termos de pinos de uma CPU ou ligações a um barramento [ADA96]. Nesse tipo de simulação, a validação temporal do modelo geralmente é necessária [SUN97], o que exige a sincronização entre os componentes de hardware e software. Os modelos de comunicação no nível RTL (*Register Transfer Level*) são considerados computacionalmente caros, devido ao grau de refinamento da interação entre os componentes [NAG2002]. Este fato motiva o uso de níveis alternativos, como por exemplo o TL (*Transaction Level*) [CAI2003].

Deste modo, os benefícios da simulação podem ser diferenciados em função do nível de abstração do modelo de representação utilizado. Modelos mais abstratos contribuem principalmente com o entendimento (observação [COS2000] e controle) e desenvolvimento das características de funcionalidade de um sistema. Modelos menos abstratos, dependendo do nível de detalhamento, podem oferecer condições para avaliação de desempenho e verificação final. Esta afirmação é derivada de uma regra básica da simulação: modelos que incorporam poucos detalhes do sistema que representam são menos realísticos, porém mais fáceis de tratar; modelos detalhados são mais realísticos, porém mais difíceis de tratar [LAW91]. Por isso, o melhor equilíbrio entre custo e benefício no uso da simulação depende dos objetivos do projetista.

Apesar do uso de diferentes níveis de abstração oferecer vantagens para as diversas etapas do processo de *co-design*, ainda existem dificuldades que precisam ser superadas. Por exemplo, o nível de abstração das interfaces, oferecido pelas linguagens de descrição de hardware tradicionais, não é suficiente para lidar com o crescimento da complexidade no projeto de sistemas e circuitos [ORT98a] [ORT98b].

Em relação aos métodos utilizados para o projeto de sistemas, a separação geralmente obscura entre comunicação e comportamento do componente dificulta a abstração das primitivas de comunicação. Por isso, é desejável a utilização de metodologias que promovam a separação entre o comportamento e comunicação nos componentes. Como exemplo, uma destas metodologias, denominada *Interface-based Design* [ROW97], permite a utilização transparente de diferentes níveis de abstração, tanto no comportamento interno do componente quanto nas interfaces de comunicação.

Alternativas que buscam a separação entre interface e comportamento adicionam facilidades para que os componentes e suas interfaces sejam inicialmente modelados em altos níveis de abstração, e sofram sucessivos refinamentos no decorrer do processo de *co-design*. O uso dessa perspectiva também permite a redução do tempo de projeto, pois a troca de nível de abstração da interface pode ser feita sem a necessidade de alterações no código do componente.

Além disso, outras abordagens como a *Selective Focus* [HIN97c] oferecem ao projetista a possibilidade de escolher o nível de detalhamento para cada um dos componentes do sistema que está sendo simulado. Deste modo, pode-se alcançar um aumento no desempenho da co-simulação na medida em que determinados componentes do sistema possam ser simulados de forma detalhada e outros, já validados ou de menor importância, em um alto nível abstração. Nesse caso, quando alguns componentes são simulados em alto nível de abstração, o descarte de alguns detalhes do modelo tende a reduzir o número de mensagens geradas pela cooperação .

O estado ótimo pode ser identificado em uma situação onde uma mesma co-simulação suporte ambos, tanto um alto grau de detalhamento quanto um bom desempenho [HIN97a]. No entanto, simulações detalhadas tendem a ser excessivamente lentas, e muitas vezes inviáveis. Por isso, técnicas que permitam a utilização simultânea de diferentes níveis de abstração (multi-nível), tanto para o comportamento quanto para a interface de componentes, tendem a otimizar o desempenho geral da simulação. Nesta alternativa, um ou mais módulos já validados são simulados com um menor número de detalhes, enquanto que módulos cujos detalhes são mais relevantes podem ser simulados em níveis mais baixos de abstração.

Por fim, investimentos em co-simulação tendem a reduzir o esforço destinado às tarefas de verificação/validação de sistemas que combinam hardware e software [SCH2003], o que justifica o desenvolvimento de pesquisas neste tema.

2.2.1 Co-simulação distribuída

Estudos em co-simulação que envolvem distribuição têm enfatizado as WANs (*Wide Area Networks*) em virtude de potenciais benefícios tais como o gerenciamento de questões relacionadas à propriedade intelectual, o desenvolvimento descentralizado de projetos, entre outros [BOR2000]. Nestes casos, os simuladores enviam e recebem dados através de interfaces de co-simulação que devem ser providas por estruturas

apropriadas que suportem requisitos de comunicação, sincronização, e tratamento de dados (tradução).

Embora a co-simulação esteja despertando maior interesse nos últimos anos, os princípios fundamentais que conduzem as pesquisas nessa área são bastante explorados na literatura. Tais princípios são discutidos em termos de conceitos/técnicas sobre comunicação, sincronização, interface, desempenho, estruturas, ambientes e modelos [CHO98a] [YOO98] [HIN97a] [YOO97] [KIM96]. Em síntese, a literatura identifica como um dos grandes desafios da co-simulação a construção de uma ponte realmente eficaz entre simuladores heterogêneos [HUB98].

Nesta perspectiva, metodologias, ambientes e ferramentas de projeto de sistemas que oferecem recursos de co-simulação têm sido propostos. Metodologias, em geral, envolvem a criação e manipulação de modelos durante o seu tempo de vida [PAG2000]. Os ambientes e ferramentas podem ser divididos em categorias de acordo com características que incorporam, como por exemplo: sistemas multilinguagens [HES2000], ambientes de projeto baseados em componentes com suporte à co-simulação [VIL2001] [RAM2000] [CHO99], ambientes de *co-design* [WAG99] [CAR2000] [OYA2000], *hardware in the loop* [STO98] [ZAN2000].

Acréscimo de complexidade pode ser observado de modo expressivo em modelos cujos componentes são distribuídos em nodos distintos de uma rede de computadores. Neste tipo de ambiente é mais difícil manter a correção em função da complexidade de aspectos relacionados com comunicação e sincronização entre tarefas [CHO99].

Outra questão relevante na simulação distribuída é o consumo de tempo em *overhead.* No entanto, a distribuição pode oferecer vantagens em relação ao desempenho em alguns casos. Em uma situação simples, tomando os componentes A e B como exemplo, se a soma dos tempos de execução de A e B é maior que a soma do tempo gasto em *overhead* e do maior componente (mais demorado), a simulação distribuída oferece condições para uma execução mais rápida. Em [COU98] são apresentados alguns experimentos que justificam esta afirmação.

Além da busca por desempenho, o tamanho de uma federação pode ultrapassar o limite de capacidade do ambiente computacional em uso numa execução centralizada. Isto torna a distribuição uma alternativa importante. Não apenas o tamanho, mas também o oferecimento de características necessárias à execução de um componente impõem conseqüências similares. Por exemplo, a existência de componentes cujas diferenças na tecnologia de construção não permite que sejam executados numa mesma arquitetura computacional. Neste caso, a distribuição também pode ser vista como a alternativa mais promissora. Questões de propriedade intelectual também podem impedir a disponibilidade de um componente no nodo onde o restante de um modelo está preparado para ser executado.

Em resumo, é verdade que o uso de ambientes distribuídos, com a sua heterogeneidade inerente, oferece maior flexibilidade aos projetistas de sistemas, por exemplo otimizando condições para projeto colaborativo, descentralização, gerenciamento de projetos maiores, incentivando o reuso de componentes, ou oferecendo (em condições plausíveis) melhor desempenho. Em contrapartida, aumentam as dificuldades para o alcance de resultados consistentes em tempo hábil em função do aumento da complexidade e das exigências pertinentes aos propósitos de um projeto, e respectivas metas. Entende-se por 'heterogeneidade inerente' como a

flexibilidade oferecida por ambientes distribuídos para que diferentes arquiteturas, técnicas, linguagens, entre outros, possam interagir sob diferentes situações/condições.

Em virtude dessa realidade, a busca de estratégias que ofereçam maior eficiência e eficácia no desenvolvimento de sistemas que agregam hardware e software (sistemas embarcados), ou sistemas heterogêneos em geral, tem motivado o desenvolvimento de pesquisas no uso da co-simulação distribuída para validação de sistemas heterogêneos.

2.2.2 Princípios de simulação distribuída

Embora a simulação distribuída tenha sido alvo de estudos há pelo menos duas décadas, atualmente a existência de melhores recursos (hardware/comunicação) a custos mais acessíveis e o aumento de complexidade dos sistemas atuais têm motivado o aumento no volume de pesquisas neste tema, por exemplo no desenvolvimento de simuladores distribuídos cujo precursor é o SIMNET (*SIMulator NETworking*) [FUL96]. Este simulador é um dos primeiros ambientes de simulação distribuída desenvolvidos na perspectiva da DIS (*Distributed Interactive Simulation*).

Na simulação distribuída, um único modelo de simulação tem suas partes (que podem ser denominadas de Processos Lógicos-PL [FER95]) executadas em ambientes computacionais distribuídos [FUJ90]. Esta característica se aplica na co-simulação distribuída. Ou seja, os mesmos princípios da simulação distribuída (principalmente relativos à sincronização) também são válidos para a execução de co-simulações com partes remotas.

A demanda por soluções paralelas e/ou distribuídas de simulação é justificada pela existência de espaço para grandes simulações em diferentes áreas, como por exemplo: ciência da computação, economia, engenharias, exercícios militares, medicina, ou qualquer outra área que possa usufruir de aplicações que consomem altas taxas de processamento.

2.2.2.1 Sincronização em simulação distribuída

A sincronização tem como objetivo geral garantir que os eventos disparados pelos Processos Lógicos ocorram em seus devidos tempos de evento (*timestamps*) [FUJ90] de modo ordenado, 'intra' e 'inter' PLs. Um evento é a alteração do valor de uma variável que representa o estado de um modelo. Cada evento ocorre em um instante do tempo de simulação, o tempo de evento. A ordem de execução de eventos internos a um PL utiliza como referência um tempo local de simulação (LVT-*Local Virtual Time*). A ordem de execução de eventos externos precisa ser controlada por um tempo global reconhecido por todos os PL's do modelo. Este tempo é conhecido como GVT (*Global Virtual Time*) [FUJ92].

Os requisitos de execução ordenada de eventos são denominados de restrições de causalidade local (LCC-*Local Causality Constraint*) [FUJ92]. Por exemplo, na Figura 2.1, o PL0 está no tempo 20 e o PL1 no tempo 60. Se PL0 envia uma mensagem para PL1 solicitando a execução de um evento no tempo 40, o estado do PL1 torna-se inconsistente. Este é um exemplo de violação de LCC [FER95].

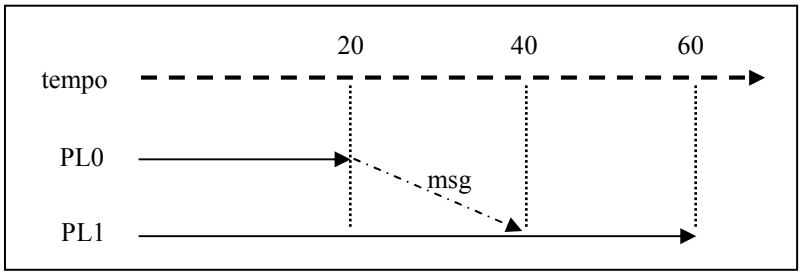

Figura 2.1: Exemplo de estado inconsistente

Pesquisas em simulação distribuída estudam propostas de algoritmos de sincronização voltados para o gerenciamento da ordem de execução de eventos. Para isso, no tratamento da sincronização, são consideradas duas abordagens (visões) distintas para a ocorrência de eventos: visão conservadora (síncrona); e visão otimista (assíncrona). No decorrer do texto, os termos 'conservador' e 'síncrono' são sinônimos, da mesma forma que os termos 'otimista' e 'assíncrono'.

2.2.2.2 Simulação Conservadora / Síncrona

Na abordagem conservadora, o protocolo que implementa as regras de controle para as trocas de mensagens deve garantir uma ordem segura de ocorrência de eventos durante uma simulação. Por exemplo, um PL(A) com LVT igual a 10 deseja executar um evento interno com tempo de evento igual a 12. Contudo, um PL(B) está preparando uma solicitação ao PL(A) para que ele execute um evento no tempo 11. Um algoritmo de sincronização conservadora deve garantir que o PL(A) execute o evento no tempo 12 somente depois que existam garantias de que nenhum outro PL irá solicitar ao PL(A) a execução de eventos num tempo menor que 12, como pretendido por PL(B) neste exemplo.

Para garantir a ordem de ocorrência dos eventos, no modo síncrono, o controle geralmente é mantido centralizado mediante o uso de um relógio global GVT (*Global Virtual Time*) [FER95]. Como na simulação síncrona os LVTs não podem ultrapassar o valor do GVT e os PLs solicitam a execução de eventos apenas para instantes de tempo superiores ao GVT, a ordem é garantida. O GVT síncrono é dado pelo maior LVT dentre os LVTs dos PLs. Deste modo, nenhum evento será executado sem garantias de que a execução de algum outro evento com tempo de evento (*timestamp*) menor possa ser requisitado. A bibliografia apresenta também algoritmos distribuídos com o mesmo propósito [FUJ2001].

Essa alternativa simplifica a implementação de simulações consistentes. Contudo, ela sub-utiliza os recursos de processamento do ambiente, pois não permite que eventos que pertençam a um intervalo de tempo futuro, e que não dependam da ocorrência de outros eventos, possam ser executados. Neste caso, para otimizar o desempenho, pode-se atribuir aos PL's a habilidade de predizer quais eventos serão gerados no futuro. Assim, também é possível determinar um intervalo de tempo futuro dentro do qual os PL's não irão gerar ou receber eventos tendo como referência o tempo do PL com menor LVT. Esse intervalo de tempo, denominado de predição de comportamento (*lookahead*) [FUJ90] [PET93], pode então ser considerado seguro.

Tabela 2.1: Mecanismos de tratamento de tempo em simulação distribuída síncrona

Mecanismo	*Função*
Lookahead	Estima período seguro de tempo futuro.
Mensagens nulas	Evita situações de impasse.
Dead-Reckoning	Redução do número de mensagens.

Embora tais mecanismos possam garantir que os eventos ocorram de modo ordenado, modelos síncronos estão sujeitos a situações de impasse (*deadlocks*). Para evitar o impasse podem ser utilizadas mensagens denominadas de mensagens nulas. Uma mensagem nula pode ser definida como uma promessa de um Processo Lógico, para os demais, de que ele não irá solicitar a execução de nenhum evento com tempo (futuro) menor que o valor fornecido na mensagem nula [FUJ2001].

Estudos em simulação distribuída também contemplam estratégias para redução do número de mensagens trocadas entre PLs. Um exemplo é o mecanismo denominado *Dead-Reckoning* [ELN2002] através do qual os PLs são habilitados a estimar o estado de outros PLs através de computações locais, reduzindo a troca de mensagens.

2.2.2.3 Simulação Otimista / Assíncrona

A abordagem otimista, em contraste com a conservadora, permite que eventos possam ser executados fora de uma ordem temporal, permitindo a ocorrência de violações de LCC. Contudo, a consistência na cooperação entre diferentes PL's é garantida através do uso de políticas de recuperação de estados consistentes em situações de erro (iminentes e ocorridas). Estas políticas, conhecidas como políticas de retorno a estados seguros, são incorporadas em protocolos de simulação distribuída que suportam execução assíncrona de eventos.

A bibliografia apresenta um conjunto de soluções bastante amplo para recuperação de estados seguros em simulação distribuída otimista. Particularmente em [ELN99] [ELN2002] é apresentada uma análise sobre as principais estratégias de retorno. Para isso, são consideradas algumas características essenciais tais como complexidade, possibilidade de ocorrência de efeito dominó, a geração de processos órfãos, entre outros.

Tomando como exemplo a Figura 2.1, o estado de PL1 torna-se inconsistente com o envio de uma mensagem de PL0 para PL1 com tempo de evento igual a 40. Neste caso, o PL1 deve retroceder no tempo até o LVT 40, tempo em que ocorreu a violação de LCC, e executar novamente os eventos até o LVT 60. As mensagens enviadas a partir de PL1 entre os tempos 40 e 60, se houverem, são denominadas mensagens órfãs. Por este motivo, a recuperação de um estado seguro pode gerar um efeito conhecido como efeito cascata. Nele, quando um PL precisa executar os eventos de um período de tempo passado, as mensagens enviadas nesse período (mensagens órfãs) podem provocar a necessidade de recuperação de estado seguro (retorno) em um segundo PL, que por sua vez pode interferir em um terceiro PL, e assim por diante [FER95].

Observa-se que a evolução de uma simulação assíncrona tende a gerar estados inconsistentes. No entanto, outras situações de erro podem ocorrer devido a outros fatores, como por exemplo, a perda de mensagens [PER2001]. Para resolver esses

problemas, autores têm proposto mecanismos que podem ser adicionados aos protocolos de controle de execução (de modelos), visando garantir consistência [FRA97], otimização/avaliação de desempenho [SPO2001] e flexibilidade [RAD98].

Salvamento de estados seguros (*checkpoints*), retorno a estados considerados seguros (*rollback*) com o uso de anti-mensagens [FER95] para indicar os eventos externos que precisam ser executados novamente e eliminação de informações não mais necessárias em atividades de retorno (*garbage collection*) são exemplos de mecanismos utilizados no gerenciamento de simulações distribuídas assíncronas. A Tabela 2.2 cita alguns destes mecanismos.

Tabela 2.2: Mecanismos de tratamento de tempo em simulação distribuída assíncrona

Mecanismo	*Função*
Checkpoints	Registro de estados seguros
Dead-Reckoning	Redução do número de mensagens
Rollback	Atividade de retorno a estados seguros
Garbage Collection	Mecanismo de identificação e exclusão de informações desnecessárias nas atividades de rollback

Em simulações otimistas, os algoritmos de retorno a estados seguros são fundamentais. Contudo, são de difícil implementação e possuem variantes que podem ser mais ou menos apropriadas de acordo com o tipo de problema alvo que se deseja solucionar com a ajuda de técnicas de simulação [ELN2002].

2.2.3 Gerenciamento do GVT

Em simulação distribuída com a necessidade de sincronização temporal é essencial que seja mantido gerenciamento sobre o GVT. No entanto, sua implementação interfere na execução da simulação na medida em que aumenta o número de mensagens e o uso de recursos de processamento dedicados ao gerenciamento do tempo. A bibliografia apresenta alternativas para gerenciamento de tempo global, algumas delas discutidas no escopo da simulação distribuída.

Como exemplo, SPEEDES GVT [STE95] é voltado para simulação distribuída e utiliza parâmetros que especificam a freqüência em que o GVT é recalculado (em tempo ou em número de eventos). Permite simulação otimista com nível de risco controlado através de parâmetros (número de eventos de risco permitidos), com o propósito de controlar o número de anti-mensagens em caso de retorno para estado seguro.

Em [LIA99] é apresentado um algoritmo para sincronização de relógio global visando processamento paralelo (sistemas concorrentes) e a coordenação de aplicações distribuídas. Este algoritmo utiliza um nodo inicial (nodo 0) que ressincroniza cada um dos demais nodos periodicamente de acordo com seu próprio relógio. A cada ressincronização o nodo inicial solicita o tempo do nodo 'N', recalcula a diferença de tempo e fornece o tempo atualizado. Este algoritmo tende a impor uma carga pesada ao sistema. Nesta mesma referência são identificados alguns outros estudos voltados para a busca e otimização de algoritmos de sincronização para tempo global.

Lamport [LAM78] apresenta um algoritmo de ordenação de eventos para uso de recursos compartilhados definido em cinco etapas: (1) a origem do evento envia uma

mensagem para todos os demais participantes com a requisição de uso de um recurso acompanhada do *timestamp*, e inclui esta requisição em uma fila de eventos local; (2) os demais participantes, ao receber a requisição, a incluem em sua respectiva fila e enviam uma mensagem de reconhecimento para a origem; (3) para liberar o recurso, a origem remove a requisição de uso da sua fila local e envia uma mensagem comunicando o tempo de fim de uso do recurso para os demais participantes; (4) os demais participantes então retiram a requisição de uso das respectivas filas; (5) para que a origem da requisição possa fazer uso do recurso duas condições precisam ser satisfeitas: o *timestamp* deve ser o menor dentre as requisições existentes na fila e a origem deve ter recebido uma mensagem de todos os demais participantes com *timestamps* maiores que o da requisição pretendida.

De modo geral, é desejável que técnicas para implementação do GVT incorporem características que promovam facilidades para escalabilidade, eficiência, suporte à interação e controle de fluxo [STE95]:

- Escalabilidade: capacidade de suportar a adição de novos componentes sem comprometimento do desempenho e consistência, mantendo o menor número possível de alterações de código e de configuração usados no gerenciamento do GVT;
- Eficiência: evitar que a atualização do GVT provoque atrasos na evolução da simulação. É desejável que o GVT seja atualizado com a maior freqüência possível sem o comprometimento do desempenho geral, gerando o menor número possível de troca de mensagens. A predição de comportamento é um instrumento que pode ser utilizado na busca deste equilíbrio;
- Suporte à interação: para simulações interativas, o GVT pode ser utilizado como referencial para bloqueio temporário da simulação, para execução do modelo em fatias de tempo controladas pelo projetista, ou para controle do disparo de eventos externos em instantes de tempo específicos;
- Controle de fluxo: Não são apenas eventos não executados e já executados que interferem nos algoritmos de atualização do GVT, mas também as mensagens ainda em trânsito. Estas mensagens não podem ser desconsideradas no cálculo de tempo global. Na solução deste problema, duas interpretações podem ser seguidas: ou o algoritmo de cálculo do GVT é informado das propriedades das mensagens em trânsito (gerando *overhead*); ou enquanto o GVT é calculado, as mensagens de simulação são temporariamente bloqueadas (uma possível fonte de ineficiência).

2.2.4 Aspectos de Projeto e Implementação de Simulação Distribuída

A complexidade da simulação distribuída exige maior esforço na construção de modelos de simulação. Além das questões de sincronização, também são exemplos de fatores que adicionam complexidade à execução distribuída de modelos de simulação:

- o trabalho de particionamento de modelos em PL´s;
- as regras de cooperação entre os PL´s (relações funcionais);

- o compartilhamento de uma porta de entrada em um PL's destino entre múltiplos PL's origem.

Em [ELN2002] [ELN99] são abordadas técnicas de projeto e implementação existentes voltadas para a construção de protocolos de simulação distribuída que prevêem o tratamento de tais fatores. Contudo, não faz parte dos objetivos desta tese discutir tais técnicas embora elas sejam utilizadas no contexto da simulação heterogênea distribuída, objeto de estudo deste trabalho.

2.3 Uso da linguagem Java em simulação distribuída

A rápida evolução da Internet, e sua natureza dinâmica, têm guiado o surgimento de novas linguagens com características distribuídas, bem como a adequação de linguagens existentes. Particularmente, a linguagem de programação Java tem sido bastante difundida para a computação baseada na Internet. Inovações observadas em linguagens como Java têm expandido o leque de possibilidades para escrever programas explorando novas oportunidades em ambientes distribuídos. Contudo, embora princípios de programação procedural ou orientada a objetos sejam hoje dominados, o conhecimento existente sobre como melhor explorar as possibilidades oferecidas pela rede de computadores ainda pode ser substancialmente expandido [PAG2000].

A linguagem de programação Java, em conjunto com as tecnologias já disponíveis para esta linguagem, tem estendido suas potencialidades para a realização de tarefas em nível de código fonte. São exemplos desta evolução os *sockets* para comunicação remota, a execução de consultas remotas em base de dados, o oferecimento de suporte para aspectos de segurança na transmissão de dados, ou a invocação remota de métodos (RMI – *Remote Method Invocation*). Em [TEO2002] é apresentado um estudo sobre o uso de JavaSpaces em simulação distribuída como instrumento de controle de uma memória compartilhada distribuída para comunicação entre processos lógicos remotos. Ainda, discute alternativas para otimização de desempenho baseada no conceito de mensagens nulas, apresentando um comparativo entre o uso de RMI e JavaSpace.

As características de Java em relação à Web incorporam capacidades relevantes no desenvolvimento de novas abordagens para modelagem e simulação [TAY99]. Por exemplo, o uso de Java facilita o desenvolvimento de ambientes de suporte ao desenvolvimento colaborativo de modelos, a execução distribuída de modelos e conseqüente suporte para a análise de resultados. Nestas atividades, a existência de alguns recursos incorporados à Java permite a redução do volume de trabalho do projetista.

Por outro lado, a evolução de linguagens de simulação de propósitos específicos também tem despertado interesse por tornar mais simples e rápido o trabalho de programação de um modelo de simulação. Contudo, tais linguagens tendem a oferecer pouca flexibilidade para modificação ou extensão de um modelo já pronto, reduzindo as chances de reuso, que hoje é uma característica altamente desejável.

Neste contexto Java também é vista como uma linguagem que permite a construção de especificações de modelos de simulação, contudo sem perder flexibilidade. Um dos exemplos é a linguagem Silk [PAG2000] de simulação que é

baseada em Java. No entanto, não faz parte dos objetivos deste trabalho o desenvolvimento de novos modelos, mas sim o suporte à execução de federações na Web.

2.4 Questões relativas ao desempenho na execução de modelos heterogêneos

Nos últimos anos, as pesquisas em simulação heterogênea estão direcionando crescentes esforços para o desenvolvimento de ambientes geograficamente distribuídos. Neste sentido, o conceito de simulação heterogênea distribuída envolve a simulação de sistemas compostos por federados heterogêneos remotos. Embora esta característica adicione flexibilidade e escalabilidade e viabilize o projeto baseado na Internet, entre outros, torna-se evidente o problema do desempenho em função da troca de mensagens via protocolos de redes de computadores.

As restrições quanto à rapidez dos meios de comunicação apresentam-se como um dos principais gargalos para a execução de modelos de co-simulação em que federados localizados remotamente precisam cooperar (respeitando os requisitos de funcionamento de cada tipo de sistema). Para tornar a co-simulação distribuída praticável com suficiente eficácia, é necessário que níveis mínimos de desempenho sejam garantidos de acordo com necessidades inerentes a cada tipo de sistema.

Alguns dos fatores que mais influenciam no desempenho de computações distribuídas (com reflexos em aspectos de sincronização) são [CHE98]:

- Velocidades diferenciadas entre múltiplos processadores;
- Variação de atrasos de comunicação;
- Falhas esporádicas de componentes (federados);
- Comportamento interativo (inerentemente não determinístico).

Os atrasos de comunicação desempenham papel dominante no consumo de tempo em comunicação, normalmente constituindo a maior porção do tempo de execução na maioria dos ambientes distribuídos [CHE98]. Por esse motivo, as tentativas de redução de custos de comunicação tendem a se concentrar na busca por implementações mais eficientes para as aplicações e também em melhoramentos no comportamento apresentado pelos mecanismos de comunicação. Implementações mais eficientes das aplicações podem, por exemplo, reduzir o número de mensagens e aumentar o tamanho de cada mensagem. Esta política tende a aumentar o desempenho.

Contudo, é verdade que a comunicação em aplicações distribuídas não pode ser completamente evitada. Mesmo assim, qualquer redução no custo de comunicação melhora o desempenho de uma aplicação, o que motiva a realização de estudos nessa linha. Em [MEL2001b] são abordadas algumas estratégias para otimização de desempenho em ambientes de co-simulação distribuída. Naturalmente, essas estratégias

podem apresentar resultados positivos em níveis diferenciados de acordo com diferentes situações de computação distribuída em que são aplicadas.

Tomando como exemplo as simulações temporizadas, a necessidade de operações de gerenciamento dos relógios de simulação interfere de forma negativa no desempenho. Isto ocorre principalmente devido ao aumento do número de mensagens em função das operações globais de sincronização [YI2003].

Como fundamento para pesquisas nessa área, algumas premissas são assumidas como verdadeiras [CHE98]. Entre elas:

- Como operações de comunicação afetam diretamente o desempenho, algoritmos distribuídos podem ser caracterizados pelo número de mensagens trocadas;
- Um número mínimo de mensagens necessárias para a realização de uma tarefa pode ser especificado. Contudo, o número máximo ou médio de mensagens varia de acordo com o ambiente (ambientes fortemente acoplados como um multiprocessador, ou fracamente acoplados como uma rede *Ethernet*);
- Diferentes padrões de protocolos de transmissão via rede podem afetar positivamente ou negativamente o desempenho da comunicação.

De modo geral, um dos principais pontos comuns às pesquisas nessa linha é inspirado na suposição, admitida como verdadeira, de que o aspecto mais promissor para se obter resultados significativos baseia-se na busca por estratégias de redução do número de mensagens transferidas entre simuladores cooperantes [KIM2002] [JUN2001].

Um segundo ponto também identificado como verdadeiro entre pesquisadores é o fato de que um ambiente distribuído de co-simulação deve ser no mínimo híbrido em relação aos modos síncrono e assíncrono de execução [YOO2000]. Ou seja, um ambiente distribuído de co-simulação onde todos os simuladores executam no modo síncrono tende a gerar um *overhead* de comunicação demasiadamente alto, potencialmente acima de níveis aceitáveis dependendo da aplicação. Por outro lado, o modo assíncrono tende a gerar menos *overhead*, contribuindo diretamente com a melhora do desempenho, mesmo apresentando o aumento do grau de complexidade como principal ponto negativo.

2.5 Abordagens e ambientes/ferramentas voltadas para a simulação distribuída heterogênea

Esta seção apresenta abordagens voltadas para co-simulação de sistemas heterogêneos, algumas de propósito geral que incluem alguma discussão sobre modelos heterogêneos, porém a maioria delas direcionadas para co-simulação, essencialmente de sistemas embarcados. Questões de distribuição são principalmente consideradas nas abordagens de propósito geral.

Esta seção dedica um espaço mais expressivo para explorar os conceitos de HLA em função de que esta arquitetura foi fonte principal para a definição das características do DCB.

2.5.1 HLA (High Level Architecture)

Desenvolvida sob a ótica da simulação interativa distribuída (DIS – *Distributed Interactive Simulation*), HLA foi criada pelo *US Department of Defense* (DoD) com base num processo de esforço conjunto envolvendo o governo, o ambiente acadêmico e a indústria. Os resultados iniciais desse trabalho surgiram em janeiro de 1995 com uma primeira definição de HLA. Em seguida, diante da continuidade dos trabalhos para o desenvolvimento dessa arquitetura, envolvendo também o *Defense Modeling and Simulation Office* (DMSO) e o *Architecture Management Group* (AMG), em setembro de 1996 o US DoD adotou HLA como padrão para suas simulações [DAH97] [DAH98].

HLA não é um padrão com um conjunto fixo de regras ou restrições. Ao invés disso, pode ser considerada como um meta-padrão que define meta-regras para construção de ambientes de simulação distribuída, respeitando as características específicas do ambiente onde estiver sendo utilizado.

A contribuição desta tese em simulação heterogênea distribuída é inspirada nas propriedades de HLA. Contudo, estende esforços para manter os federados sob o conceito de *black-box* [WAG94].

2.5.1.1 Aspectos técnicos de HLA

Em HLA, o projeto de sistemas de simulação incorpora o conceito de federados [NAN99]. Os modelos federados podem ser considerados como uma extensão do conceito de objetos, originado nos anos 60 com a linguagem de simulação SIMULA [NYG81]. A linguagem SIMULA permite que objetos sejam gerados através da instanciação de classes, herdando os atributos e métodos (operações).

Uma federação é a combinação de FOMs (*Federation Object Model)*, de um conjunto de federados e da RTI (*Run-Time Infrastructure Services*) [FUJ96]. FOM é formado por um conjunto de Modelos de Objetos de Simulação (SOM – *Simulation Object Model*) [KUH2000]. A RTI é um sistema que visa o controle das operações distribuídas (troca de mensagens), sendo sua aplicação direcionada para o conceito de federação.

Um federado pode representar um modelo de simulação em computador, um simulador de propósitos específicos, um serviço de interação com a simulação como um todo, ou mesmo uma interface para um participante físico/vivo da simulação. A Figura 2.2 apresenta uma visão funcional de uma federação HLA.

Um federado pode ser visto como uma entidade ou componente de um modelo. Um conjunto de federados cooperantes, sob o controle de determinadas regras, forma uma federação. O uso deste conceito permite que o trabalho de modelagem de um sistema utilize componentes modulares, o que oferece melhores condições de organizar adequadamente a definição de interface e funcionalidade.

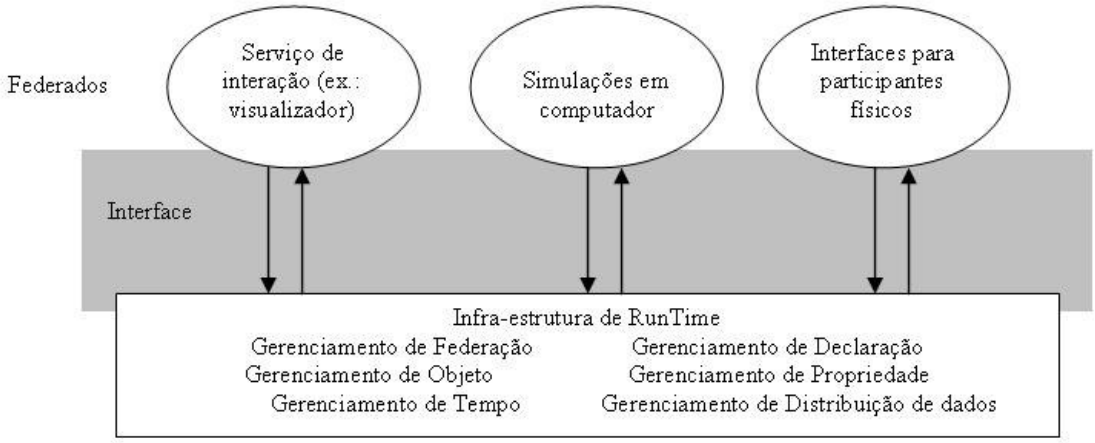

Figura 2.2: Visão funcional de uma federação HLA

Não são impostas restrições sobre 'o que', e 'como', é representado nos federados. Contudo, todos os federados precisam incorporar características que os permitam interagir com objetos de outras simulações através da troca de dados. Esta troca de dados é suportada pela RTI [CAR97].

HLA possui um conjunto de componentes. O primeiro componente é definido pelo conjunto de regras que descrevem as responsabilidades/atribuições das simulações (responsabilidades da RTI). Ele é dividido em regras de federação e regras de federados. O segundo componente é a especificação de interface HLA, e o terceiro, o *Object Model Template* (OMT). O OMT visa especificar um formato comum e estruturado para documentação dos modelos de objetos HLA.

São **regras de federação** [BUS98]:

- Federações devem ter um FOM no formato OMT;
- Toda representação de objetos deve estar nos federados e não na RTI;
- Durante a execução da federação, toda troca de dados entre FOMs deve ser via RTI;
- Durante a execução da federação, todo federado deve interagir com a RTI de acordo com a especificação da interface;
- Durante a execução da federação, um atributo de uma instância de um objeto pode ser propriedade de somente um federado em um dado tempo.

São **regras de federado**:

- Federados devem possuir um SOM no formato OMT;
- Todos os federados devem ser capazes de atualizar/repassar atributos e enviar/receber dados de acordo com seus SOMs;

- Federados devem ser capazes de transferir/aceitar atributos próprios de acordo com seus SOMs;
- Federados devem ser capazes de variar as condições sob as quais devem fornecer atualizações de atributos de acordo com seus SOMs;
- Federados devem ser capazes de gerenciar o tempo local de modo que permitam executar a troca de dados com outros membros da federação.

A especificação de interface, componente número dois de HLA, tem o objetivo de padronizar a interação entre federados e a RTI. Este componente é visto como uma API (*Application Programmer Interface*) em diferentes formatos, tais como CORBA IDL, Java, C++, etc.

A RTI possui seis serviços básicos (Figura 2.2): gerenciamento da federação, gerenciamento de declaração, gerenciamento de objeto, gerenciamento de propriedade, gerenciamento de tempo, e gerenciamento de distribuição de dados. Estes serviços têm as seguintes funções:

- gerenciamento de federação: utilizado para criação, controle dinâmico, modificação e eliminação da execução de uma federação;
- gerenciamento de declaração: divulga as intenções da federação de acionar (publicar e subscrever) interações e atributos de objetos;
- gerenciamento de objeto: visa o tratamento do registro, modificação e eliminação de instâncias de objeto, bem como enviar e receber interações de objetos;
- gerenciamento de propriedade: é utilizado por federados para transferir a propriedade de atributos de instâncias. Isto permite a modelagem cooperativa na federação;
- gerenciamento de tempo: controla o avanço do tempo lógico, fazendo com que as informações sejam tratadas sem transgressões de causalidade;
- gerenciamento de distribuição: aborda o trabalho de vigilância sobre dados que são recebidos e transmitidos com o objetivo de reduzir a troca de informações consideradas irrelevantes.

O componente *Object Model Template* (OMT) é uma estratégia utilizada para tratar a informação contida no modelo de objeto HLA para cada federação (FOM) e para cada simulador (SOM). Modelos de objetos visam a descrição de detalhes considerados críticos, tanto das federações como das simulações. HLA não impõe restrições no conteúdo dos modelos de objetos, no entanto, exige que a documentação de federados e federações utilizem o padrão OMT para que possuam uma interface compatível com a RTI.

Ambos, FOMs e SOMs, são tipos de modelos de objetos especificados por HLA, e são documentados segundo o OMT. O FOM descreve o conjunto de objetos, atributos e interações pertinentes à federação. O SOM descreve o federado (simulador)

considerando o tipo dos objetos, atributos e interações que podem oferecer para federações.

2.5.1.2 Benefícios e algumas limitações no uso de HLA

De acordo com as características da arquitetura HLA, algumas questões podem ser abordadas em função dos benefícios que podem ser alcançados, bem como de algumas deficiências [BUS98] [DAH97] [TOL2002]:

- HLA utiliza APIs, contudo a comunicação entre federados é de responsabilidade da RTI que ele utiliza, o que pode tornar mais complexa a implementação de federações uma vez que a RTI incorpora também todo o gerenciamento dos itens de simulação;
- Como HLA é especificamente direcionada para simulação distribuída, o suporte oferecido por essa arquitetura para aspectos específicos de simulação facilita sua implementação (por exemplo, o gerenciamento de tempo);
- HLA permite oferecer e fazer uso de serviços, contudo não suporta comunicação direta entre objetos;
- O controle sobre a propriedade de objetos adiciona flexibilidade para modelar alguns tipos particulares de simulações;
- Como HLA não especifica um protocolo de comunicação em particular, diferentes implementações de RTIs podem ser surpreendidas com problemas de comunicação, se implementadas com diferentes padrões de comunicação;
- Como HLA requer que a interoperabilidade entre federados seja construída sobre a RTI, e a troca de dados feita via interfaces específicas, a arquitetura mantém efetivo compartilhamento de dados e sua conseqüente interpretação de forma consistente. Contudo, reduz a flexibilidade para reuso de federados;
- A imposição de moldes para a construção de federados limita a liberdade do projetista, bem como o reuso de componentes existentes como um novo federado em uma federação. Além disso, exige consideráveis modificações/inclusões no código para a chamada de primitivas de serviços da RTI. Ou seja, não há preservação do código original de um componente que precisa ser incluído numa federação;
- Todos os serviços de comunicação e gerenciamento da simulação são atribuídos à RTI, o que pode resultar em aumento de complexidade para sua implementação.

A necessidade de incluir operações de gerenciamento da federação no código do federado (através de chamadas à primitivas da RTI) compromete sua integridade. São exemplos de chamadas a primitivas e respectivos propósitos:

- *public hla.rti.FederateHandleSetFactory _federateHandleSetFactory* : declaração dependente da implementação em uso da RTI;
- *_rti = RTI.getRTIambassador(...)*: declara o uso explícito, pelo federado, de um embaixador definido na implementação da RTI em uso;

- *_rti.createFederationExecution(...) / _rti.joinFederationExecution(...)*: pedido de execução de um federado seguido da sua inclusão em uma federação;
- *_rti.enableTimeConstrained() / _rti.enableTimeRegulation(...) / _rti.synchronizationPointAchieved(...)*: tratamento de restrições para sincronização no tempo;
- *_rti.getAttributeHandle(...)*: tratamento de atributos;
- *_rti.attributeOwnershipAcquisitionIfAvailable(...)*: federado verifica disponiblidade de atributo;
- *_rti.negotiatedAttributeOwnershipDivestiture(...)*: federado solicita proprieade sobre um atributo;
- *_rti.updateAttributeValues(...)*: tratamento de valores de atributos;
- *defaultRTIport = "..."* : define a porta de comunicação com o servidor RTI central;
- *_rti.getObjectClass(...)*: utilizado para definir um objeto em uso.

Estes são alguns exemplos de primitivas RTI que precisam ser utilizadas explicitamente pelo federado para que ele possa participar de uma federação. As primitivas citadas como exemplo são utilizadas na implementação de uma RTI disponível em [KHU2000]. Em resumo, as operações de comunicação ou sincronização executadas por um federado exigem o uso de primitivas implementadas na RTI em uso e o conhecimento das propriedades da federação por parte do federado (p.ex. porta de comunicação do componente central da RTI). Esta característica adiciona complexidade e limitações no reuso e interoperabilidade, essencialmente na representação de sistemas heterogêneos, proporcionalmente ao número de operações de gerenciamento que precisam ser incluídas no federado.

Afora as exigências sobre os federados, e mesmo provendo um padrão auto-suficiente para a implementação de simulações distribuídas, alguns estudos propõem sua interconexão com outras plataformas de simulação com o objetivo de usufruir das vantagens específicas de cada uma delas. Dentro desta perspectiva, foi apresentada em [ZEI99b] a implementação de um protocolo de simulação, denominado DEVS (*Discrete Event System Specification*) [CHO94] [ZEI99a], para simulação baseada em evento que explora recursos de HLA. Essa interconexão permite o uso de uma simulação baseada em eventos (característica de DEVS) sobre um ambiente distribuído (suportado pela RTI).

2.5.2 HLA e MDA (*Model Driven Architecture*)

O aparecimento de dificuldades no uso de HLA foi percebido na medida em que esta arquitetura despertou interesses não militares e não se tornou um *backbone* de simulação difundido como previsto. Um dos fatores mais significativos é a proliferação de diferentes implementações da RTI sem que federados construídos para uma RTI sejam compatíveis com as demais. Entre outras conseqüências, este fato compromete o reuso de federados até mesmo entre diferentes implementações do próprio padrão HLA.

Com o objetivo de reverter esta tendência, a integração dos conceitos de MDA (*Model Driven Architecture*) em HLA tem sido discutida no contexto da modelagem e simulação. MDA prega a idéia de meta-modelagem de forma que padrões distintos (p.ex. CORBA, XML [STY2001], EJB, DCOM), geralmente utilizados separadamente, sejam aplicados em meta-modelos com objetivos comuns [TOL2002].

O uso dos conceitos de MDA em HLA implica, principalmente, na forma como os serviços de simulação são fornecidos e nas propriedades da RTI. Os serviços fornecidos pela RTI/HLA devem estar em harmonia com os serviços oferecidos pelos demais padrões integrados às atividades de modelagem e de simulação, mesmo que sua essência (técnicas de suporte à simulação) seja mantida por HLA. Por exemplo, as primitivas RTI devem possuir afinidade com serviços CORBA, DCOM, EJB, etc, se utilizados.

Em relação à RTI, suas funcionalidades devem ser definidas de acordo com um modelo de aplicação independente de plataforma. Deste modo, a RTI seria transformada em uma solução de integração com princípios similares a padrões tais como CORBA, contudo oferecendo serviços específicos de modelagem e simulação. Ou seja, não apenas o modelo conceitual de HLA, mas também sua implementação, estariam submetidos a formatos específicos. Em relação aos federados, ao invés de reconhecer os serviços de uma implementação particular da RTI, eles devem reconhecer a interface RTI definida pelo uso conjunto de MDA e HLA. Esta característica mantém restrições de interface de comunicação, o que não é desejável na simulação heterogênea.

MDA, no entanto, não resolve diretamente problemas de gerenciamento de dados no desenvolvimento de tarefas de tradução na comunicação entre federados quando possuem interfaces incompatíveis. Em [TOL2002] a discussão sobre o uso de MDA em conjunto com HLA permanece no escopo teórico.

2.5.3 Co-simulação Multilinguagem

Abordagens multilinguagens são utilizadas para o projeto de sistemas formados por componentes heterogêneos através do oferecimento de suporte para a interação entre múltiplas, porém limitadas, linguagens de descrição de sistemas. Na especificação de sistemas heterogêneos com o uso dessa abordagem, cada parte do sistema pode ser descrita através de uma linguagem diferente que melhor se adequa às características de cada uma destas partes. O uso de múltiplas linguagens também permite que o projeto seja descrito por diferentes equipes e em diferentes níveis de abstração, e cada equipe pode também utilizar seu próprio método de trabalho. No entanto, embora a validação individual das partes não seja comprometida, torna-se mais difícil validar o sistema inteiro quanto à cooperação entre as diferentes partes do sistema, ou sub-sistemas. Esta dificuldade existe, principalmente, devido aos aspectos de coordenação e sincronização na troca de dados entre os sub-sistemas [ERN2000] [HES99].

Estas abordagens geralmente suportam a co-simulação e a síntese de sistemas que integram hardware e software. Um exemplo de ferramenta que incorpora esta abordagem é a MCI (*Multilanguage Cosimulation Interface*) [HES98]. Ela inicia de uma descrição abstrata de uma interface de comunicação para a conexão de múltiplos sub-modelos e automaticamente gera um ambiente de co-simulação especializado para a respectiva aplicação em estudo.

A comunicação entre os sub-modelos é mantida por um barramento de co-simulação e por uma interface de simulação. Esta interface, em MCI, é mantida por uma unidade de comunicação que possui procedimentos específicos para a troca de dados entre sub-modelos. Para isso, a maior dificuldade está no tratamento da heterogeneidade dos tipos de dados de interface dos sub-sistemas. Também existe a limitação no número de linguagens suportadas, embora outras linguagens possam ser adicionadas (o que exige manutenção na ferramenta).

MCI não utiliza sincronização entre os simuladores, embora cada simulador mantenha um relógio local. Sem sincronização, as trocas de dados entre simuladores são controladas através dos eventos de forma que apenas a ordem de execução das ações seja mantida. Deste modo, uma co-simulação baseada em MCI permite observar o funcionamento do modelo demonstrando a cooperação entre sub-modelos.

A ferramenta permite executar uma especificação multilinguagem em paralelo em uma estação, ou em uma rede alocando cada sub-modelo a um nodo distinto.

2.5.4 O ambiente WESE

WESE (*Web-based Environment for Systems Engineering*) é um ambiente colaborativo de desenvolvimento de sistemas baseado na Web [RAO2000a]. Motivado pelo potencial da infra-estrutura WWW para troca de informações em grande escala, WESE é o resultado de estudos baseados na busca de soluções para problemas relacionados à engenharia de sistemas. Entre eles, são endereçados pelos autores aspectos de disponibilidade, acessibilidade, interoperabilidade e validação de sistemas através da simulação [WIL99]. Neste ambiente, o suporte à simulação distribuída é fornecido por um simulador de eventos discretos paralelo denominado WARPED [RAD98]. Este simulador utiliza o mecanismo *Time Warp* para as tarefas de sincronização e o conceito de Processos Lógicos (PLs) para a representação de componentes modelados.

Este ambiente também permite que o projetista de sistemas utilize técnicas formais (métodos formais) para verificar e validar os modelos. Para isso, oferece infra-estrutura para desenvolver especificações de modelos na linguagem PVS (*Prototype Verification System*) [CRO95] que permite a prova formal de especificações (através de teoremas).

Na construção e execução de uma simulação, o primeiro requisito do ambiente é a construção de uma descrição do sistema a ser simulado em uma linguagem denominada SSL (*System Specification Language*) projetada no escopo do WESE [RAO2000a]. O fonte SSL é analisado de acordo com um código intermediário denominado SSL-IF (*Intermediate Form*) que monta a entrada para outros módulos de WESE. SSL-IF é utilizado para construir simulações, gerar especificações e documentações de *factories* distribuídas. As *factories*, que são repositórios distribuídos de componentes, desempenham um papel fundamental no provimento de uma interface uniforme para gerar os componentes especificados em SSL. Para interagir com o ambiente, os usuários podem usar HTML (*Hyper Text Markup Language*) ou uma interface baseada em texto.

O mecanismo WESE de entrada e saída é constituído de uma CGI (*Common Gateway Interface*) baseada em HTML, de rotinas de interface de texto, e de um *parser* SSL. As interfaces baseadas em texto ou em HTML são utilizadas para interação com o

ambiente. Os CGIs são agregados às páginas HTML com o objetivo de fornecer a interatividade necessária para que o WESE seja acessado via *browsers*.

Os arquivos fonte SSL representam os modelos de um sistema qualquer em estudo, e eles consistem de módulos interconectados, devendo ser inicialmente fornecidos ao WESE antes de uma simulação. Cada descrição SSL é dividida em três seções principais:

- *Component definition section*: contém detalhes dos componentes que são utilizados para especificar os módulos;
- *Component instantiation section*: define os componentes que constituem um módulo;
- *Netlist section*: define a interconectividade entre múltiplos componentes instanciados.

SSL permite que diferentes componentes sejam especificados para um mesmo sistema. Para isso, o usuário precisa conhecer a localização das *factories* e a descrição dos componentes que podem ser gerados por cada *factory*. Os desenvolvedores das *factories* devem fornecer as informações necessárias (interface, detalhes de implementação, etc). Então, a partir de uma entrada SSL, é obtido um SSL-IF (*Intermediate Form*). O SSL-IF (que promove uniformidade de interface) é utilizado pela seção de montagem (*Elaborator)* para compor simulações, abrir as especificações e gerar relatórios sobre o sistema modelado.

A construção e execução de modelos é realizada através de módulos que combinam um gerente de simulação, um gerente de informação, um gerente de *factory* e *factories* distribuídas. O gerente de simulação fornece a funcionalidade necessária para compor um modelo simulável a partir de *factories* distribuídas. O SSL-IF é utilizado nesta tarefa para permitir que sejam colecionados os módulos de simulação necessários. Utilizando o gerente de *factory*, o gerente de simulação interage com *factories* distribuídas para construir os objetos necessários para a simulação. É obrigação do gerente de *factory* oferecer uma interface adequada para a interação com *factories* distribuídas. Por fim, o gerente de informação desempenha a tarefa de colecionar informações sobre os componentes de acordo com os requisitos determinados pelo usuário.

Para o estabelecimento de comunicação com componentes disponíveis em uma *factory* existe um módulo denominado *gateway*. Este módulo estabelece comunicação entre endereços especificados em função de requisições de usuários (através de gerentes de simulação) utilizando um *backbone* de comunicação. O *backbone* oferece uma interface para a infra-estrutura de comunicação utilizada na Web. Assim, múltiplos gerentes de simulação distribuídos podem se comunicar com uma mesma *factory* através do módulo *gateway*. Um *gateway* não mantém a comunicação com cada componente de acordo com sua interface, mas sim a comunicação com os componentes de uma *factory* através de uma interface padrão de acordo com as descrições SSL.

A comunicação é suportada por módulos que fornecem um *backbone* base de comunicação que, em conjunto com o subsistema de simulação, forma o que é denominado de subsistema de comunicação (Figura 2.3). A construção deste subsistema é feita sobre *sockets* TCP (*Transmission Control Protocol*).

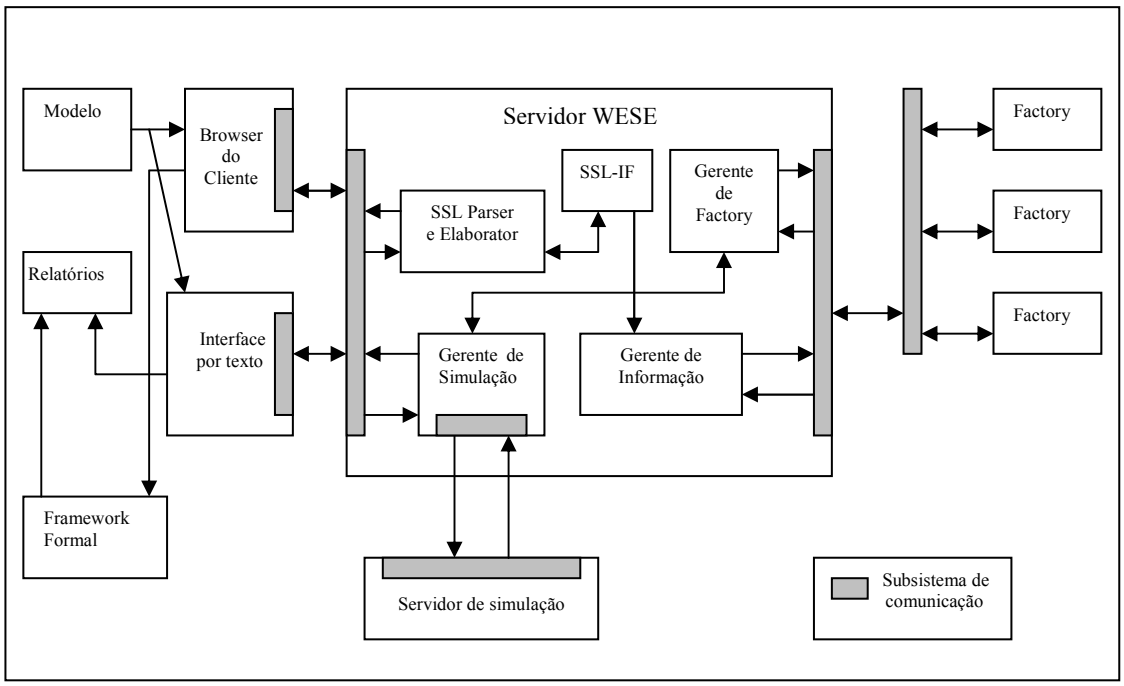

Figura 2.3: Visão geral do WESE

Ao ambiente WESE foi incorporado o DCS (*Dynamic Component Substituition*). O DCS é um módulo que permite a substituição de componentes durante a simulação. Contudo, a implementação do DCS no ambiente WESE interfere na estrutura da SSL, da SSL-IF e na infra-estrutura de simulação. Em [RAO2000b] a estrutura que viabiliza a DCS com o uso do WARPED é abordada em detalhes.

2.5.5 A ferramenta IPCHINOOK

IPCHINOOK [CHO99] é uma ferramenta de projeto de sistemas embarcados distribuídos que permite a execução de co-simulações. Para a construção de um projeto é utilizada uma interface com usuário fornecida pelo simulador *Pia* [HIN97b].

Na construção de um projeto, inicialmente deve ser fornecida para a ferramenta uma descrição das funcionalidades (comportamento) do modelo em desenvolvimento que especifica os módulos que irão interagir durante uma co-simulação. Estes módulos são definidos como *modal processes* [CHO98b]. Também deve ser fornecida uma descrição da arquitetura do hardware pretendido (processadores, dispositivos de I/O, barramento de comunicação, topologia). Por fim, são definidas funções de alocação entre ambas as descrições. Estas funções determinam, por exemplo, em qual processador um *modal process* irá executar.

Para coordenar a comunicação entre componentes previstos nas descrições, estes devem reconhecer um protocolo comum. Caso isto não seja possível, a ferramenta oferece um conjunto (biblioteca) de primitivas denominado ACT (*Abstract Control Types*) que visa permitir uma adaptação que viabilize a composição desses componentes. A possibilidade de reuso de um módulo é dependente da sua compatibilidade com as primitivas de ACT.

A ferramenta não automatiza o particionamento de modelos nem o modo como módulos são interligados entre si. Estes itens devem ser definidos e fornecidos pelo projetista do modelo.

IPCHINOOK também implementa o conceito de foco seletivo (*selective focus*) definido no escopo das ferramentas *Pia* [HIN97c].

2.5.6 Projeto *Ptolemy*

O projeto *Ptolemy* [LEE2001a] pode ser definido como uma estrutura para realizar a interação entre componentes de um sistema. Tendo como foco principal o projeto de sistemas embarcados, os esforços deste projeto estão direcionados para estudos em modelagem heterogênea, simulação, e projeto de sistemas concorrentes. Sucedendo o *Ptolemy Classic* [HIN97b], o *Ptolemy II* é a segunda geração deste projeto. O *Ptolemy II* apresenta uma outra forma de estruturação e envolve o uso de Java, de concorrência e de integração com redes de computadores.

No Ptolemy II a construção dos modelos de simulação é realizada através da especificação de conjuntos de regras que governam as interações entre os componentes dos modelos e dos modelos com o mundo externo [LEE2001a].

Para manter o controle sobre a variedade e quantidade de componentes, é utilizada uma estratégia de contenção dentro de ambientes denominados *polymorphic domains*. Esta estratégia capacita a interação entre componentes que podem estar em uma grande variedade de domínios distintos. Ou seja, ao invés de projetar mecanismos de interação diferentes, os componentes é que são projetados para interagir de diferentes formas, tornando-os mais reusáveis. Para isso as interações precisam ser disciplinadas por uma semântica bem definida. No caso de reuso de componentes existentes é sugerida em [LEE2001a] a construção de adaptadores entre interfaces incompatíveis ou o uso de técnicas baseadas em polimorfismo para tratar a compatibilidade de interfaces e verificação de tipos.

Existem modelos de computação que tratam de modos distintos as questões de concorrência e de tempo na cooperação entre os componentes. No *Ptolemy* estes modelos de computação são voltados para sistemas embarcados.

Os modelos de computação são:

- *Communicating Sequential Processes* (CSP)
- *Continuous Time* (CT)
- *Distrete-Event* (DE)
- *Distributed Discrete Event* (DDE)
- *Discrete Time* (DT)

- *Finite-State Machines* (FSM)
- *Process Networks* (PN)
- *Synchronous Dataflow* (SDF)
- *Synchronous/Reactive* (SR)

Uma descrição individual de cada um destes modelos de computação é apresentada em [LEE2001a].

2.5.7 Ferramentas Pia

As ferramentas de co-simulação *Pia* foram desenvolvidas como um domínio do projeto *Ptolemy Classic* com o objetivo de representar a troca de dados entre simuladores distintos, permitindo também que estes dados sejam alterados em tempo de simulação. Para descrever a funcionalidade dos modelos simulados estas ferramentas utilizam uma linguagem específica própria.

As ferramentas *Pia* fornecem um mecanismo de especificação que permite a interconexão de nodos através de uma rede de comunicação, podendo incluir nodos distribuídos geograficamente. Os nodos são interconectados com o uso de *sockets*, viabilizando a conexão com outras ferramentas de projeto bem como outros simuladores, ou compiladores, ou mesmo dispositivos. No entanto, estas ferramentas precisam ser construídas com suporte para *sockets* no formato definido em *Pia* [HIN98].

Ferramentas *Pia* permitem alterar dinamicamente (em tempo de execução de uma simulação) o nível de detalhamento representado por um simulador. Para isso, o simulador precisa estar preparado para atender as respectivas solicitações de alteração de nível. Esta funcionalidade é denominada de foco seletivo (*selective focus*) [HIN97c], e permite, num mesmo modelo, a composição de módulos ditos de alto nível e módulos de baixo nível. *Pia* também possui mecanismos de tratamento de tempo (sincronização).

Dentre as principais características do domínio *Pia*, destacam-se: o não uso de filas de eventos; construções de suporte para *checkpoint* e para ações de recuperação; e construções de suporte para gerenciamento do tempo.

Sistemas simulados sobre *Pia* consistem de quatro tipos de objetos. Os tipos de objetos são os seguintes [HIN97]:

- Componentes: são coleções de interfaces e comportamentos (que podem incluir software embarcado). Eles podem ser utilizados para representar componentes físicos ou componentes virtuais na visão de uma especificação;
- Interfaces: são coleções de portas, rotinas de *drivers*, e tratadores de eventos;
- Portas: são emissores ou receptores de dados e eventos. Elas são utilizadas para comunicação com componentes externos;
- Redes: são objetos para conexão de portas.

2.5.8 A ferramenta JavaCAD

A ferramenta JavaCAD [DAL99] implementa uma proposta de simulação distribuída com o uso de componentes IP. JavaCAD é implementado como um *backplane* de simulação e suporta avaliação de componentes IP durante a especificação do projeto, enquanto também visa a proteção tanto para o usuário bem como para o fornecedor dos componentes. Com isso, é possível instanciar componentes IP de múltiplos fornecedores remotos e simulá-los junto com blocos proprietários.

No lado do fornecedor do componente existe preocupação quanto à proteção da propriedade intelectual. Para isso, é dada ao fornecedor autonomia para decidir se um componente pode ou não ser utilizado por terceiros sem comprometimento de informações confidenciais. Informações sobre a intenção de uso do componente precisam ser previamente disponibilizadas ao fornecedor.

O JavaCAD é composto por um conjunto de pacotes (escritos em Java) denominado *JavaCAD Foundation Packages* (JFP). Para que uma simulação virtual (baseada em *backplane*) possa ser executada, os fornecedores de componentes IP e os usuários precisam utilizar o JFP. Deste modo, componentes simples, ou mesmo uma coleção de componentes interconectados, são instanciados como subclasses das classes básicas de JFP que contém um simulador multi-nível dirigido a evento.

Essa ferramenta também fornece suporte para simulações concorrentes de diferentes partes de um projeto (ou diferentes configurações de um mesmo projeto) com base em múltiplas linhas de execução (*threads*).

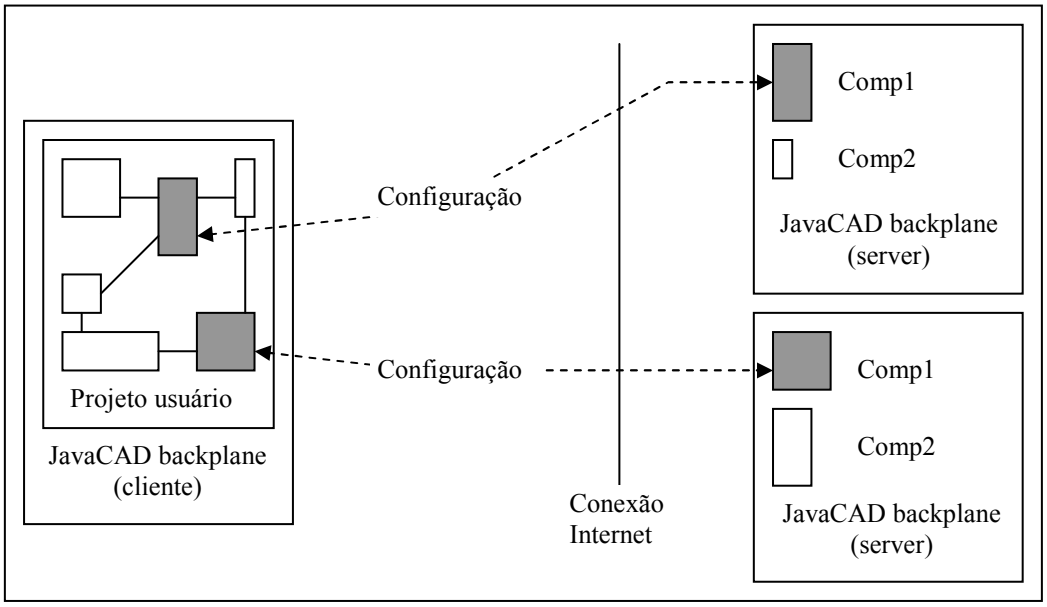

Figura 2.4: Arquitetura do ambiente de projeto de JavaCAD

Para a realização de uma simulação, inicialmente o usuário deve especificar o projeto através de um cliente JavaCAD. O projeto pode conter componentes IP de um ou mais fornecedores. Para que os fornecedores possam se comunicar com os clientes,

são utilizados servidores JavaCAD. No processo de configuração da simulação, inicialmente ocorre uma negociação (entre cliente e servidor) para que seja estabelecida a funcionalidade e o custo de execução dos componentes que serão adotados na simulação. Feita a configuração, a simulação pode ser executada, e cabe ao *backplane* manter a comunicação entre clientes e servidores através da Internet. Como resultado, tem-se um ambiente de simulação onde um ou vários componentes são virtuais. A Figura 2.4 apresenta a arquitetura conceitual do funcionamento de JavaCAD.

Em JavaCad, todo componente é uma subclasse de uma classe denominada *Module*. Uma instância da classe *Module* é especializada por um conjunto de métodos que são executados quando ocorrem eventos que devam ser tratados pelos respectivos componentes e um conjunto de conexões, sendo que cada conexão é atribuída a um *Connector* diferente. A partir dessa classe, o comportamento do componente pode ser especificado em diferentes níveis de abstração.

Os *Modules* de um projeto podem ser locais ou remotos. Um *Module* local sempre executa na JVM (*Java Virtual Machine*) do usuário dos componentes IP. Um *Module* remoto é geralmente executado no servidor JavaCAD do respectivo fornecedor do componente IP.

O objetivo do *Connector* é de manter a troca de mensagens entre *Modules*. Não existe outra forma de conectar dois *Modules* a não ser através de um *Connector*, que é responsável pelo tratamento da comunicação e que representa conexões ponto-a-ponto. Deste modo, conectores mais complexos podem ser projetados para situações não triviais de comunicação sem que haja comprometimento dos *Modules.*

O recurso Java RMI (*Remote Method Invocation*) é utilizado para a implementação das classes *Modules* remotas. A possibilidade de implementar *Modules* remotamente não significa uma exigência. Este recurso é empregado principalmente para os casos onde os métodos contêm informações sujeitas a restrições de propriedade intelectual, sendo necessário o estabelecimento de segurança e proteção.

O problema da segurança na comunicação entre um cliente e um servidor em relação a intrusos (acesso não autorizado) é resolvido pela implementação Java do protocolo RMI. Para a proteção de fornecedores IP, JavaCAD implementa a especificação dos *Modules* remotos em duas partes. A parte IP protegida da especificação do componente é localizada no fornecedor IP como uma classe privada (*private class*) cujo *byte-code* não é enviado para o cliente. A segunda é a parte considerada pública, também pertencente à especificação do componente, mas que é enviada ao usuário IP pelo seu fornecedor. Então ela é utilizada pelo usuário IP para instanciar o componente remoto dentro do projeto. Ou seja, a parte pública de um *Module* executa as atividades necessárias para a invocação de métodos remotos que irão executar no lado do servidor mantido pelo fornecedor do componente IP.

Por outro lado, JavaCAD também implementa a proteção da propriedade intelectual sobre o projeto do usuário, enquanto ele usufrui da funcionalidade de componentes remotos. Este nível de proteção é mantido através da imposição de restrições no tratamento dos parâmetros (envio de mensagens). O modo como se dá a relação funcional entre *Modules* e *Connectors* permite inibir a transmissão de informações consideradas sigilosas.

2.5.9 A plataforma de simulação CORSA

CORSA (*Component-ORiented Simulation Architecture*) [CHE2001b] é uma plataforma de simulação baseada numa visão de mundo orientada à componentes. A comunicação entre componentes se dá através de portas, e não pela ocorrência de eventos. Com isso, CORSA busca interoperabilidade entre componentes, beneficiando também o reuso. Esta plataforma prevê componentes de três tipos em relação ao tempo: não temporizados, que consideram o tempo mas sem autonomia para avanço, e componentes síncronos.

Componentes não temporizados são sensitivos às suas portas de entrada. Ou seja, apenas enviam mensagem mediante uma ocorrência numa porta de entrada. O segundo tipo percebe a existência de tempo, mas precisa solicitar seu avanço para a plataforma de simulação. O terceiro tipo é síncrono, como definido em [FER95]. Ou seja, só pode receber mensagens com tempo de evento maior que o tempo global.

CORSA define um padrão para o desenvolvimento de componentes que pode ser utilizado como um contrato entre desenvolvedores de máquinas de simulação e de componentes. Simuladores não projetados de acordo com o modelo CORSA não são suportados. Componentes assíncronos também não são previstos, e a plataforma é direcionada para execução centralizada (não suporta distribuição dos componentes).

2.5.10 Ferramentas comerciais

Ferramentas comerciais utilizadas no projeto de sistemas embarcados também incorporam recursos de co-simulação para validação de sistemas. São exemplos de empresas que distribuem ferramentas de co-simulação: *Coware* [COW2003], *Cadence* [CAD2003], *Mentor Graphics* [SEA2003], *Synopsys* [SYN2003]. No entanto, em geral, as soluções de co-simulação nelas disponíveis são específicas para os recursos de projeto de sistemas eletrônicos incorporados às ferramentas.

Coware possui ferramentas para a depuração de componentes de software e de hardware que permitem a execução de coverificação em tempo de projeto. *Cadence* oferece um módulo de co-simulação denominado *Incisive Unified Simulator* que suporta Verilog, VHDL e SystemC. Para a execução de uma co-simulação este módulo gera um bloco de código específico. A *Mentor Graphics* possui uma solução para coverificação denominada *Seamless*. Ela permite a co-simulação entre um módulo de software e uma descrição de hardware implementada com o uso de *ModelSim*.

Synopsys possui um ambiente de projeto em nível de sistema denominado *System Studio* (SS) que oferece condições para o uso de recursos de co-simulação. Para isso, descrições em VHDL ou Verilog em *Synopsys* VCS/VCS MX são combinadas com C++ no *System Studio* através da DKI (*Direct Kernel Interface*). SS também pode interagir com *ModelSim* e *NC-Sim* e importar modelos de *Matlab*. Se uma simulação for controlada por SFC (*Simulation Control Files*), ela pode ser distribuída sobre estações interligadas com o uso da plataforma LSF (*Load Sharing Facility*).

Em geral, os recursos de co-simulação disponíveis numa ferramenta comercial são dimensionados para ajudar na solução de problemas apenas no escopo das alternativas de projeto de sistemas embarcados previstos pela ferramenta.

2.6 Análise comparativa

Esta seção apresenta uma breve análise comparativa entre as ferramentas/metodologias comentadas neste capítulo. Esta análise tem como objetivo explorar as perspectivas e possibilidades para a simulação heterogênea. Ela é fundamentada nos desafios atuais, citados na literatura, e relacionados com a busca de estratégias eficazes para a integração de partes (ou componentes) heterogêneas na construção de modelos de co-simulação, e também no suporte à distribuição física desses modelos em tempo de execução.

HLA, embora seja uma arquitetura que ofereça vantagens para o gerenciamento de federações, possui aspectos restritivos como discutidos neste capítulo. Com maior ênfase a partir de meados de 2002, algumas publicações têm iniciado discussões sobre o processo de maturação desta arquitetura. Tanto a necessidade de melhorias como a restrição de HLA ao domínio militar são citadas como justificativa para o fato de que, em geral, as tentativas de utilizar HLA fora do domínio militar não têm tido o sucesso esperado. Afirmações deste tipo têm surgido em publicações mais recentes, onde têm sido divulgadas contribuições para possíveis soluções às deficiências [TOL2002]. Tais propostas podem ser divididas em duas categorias: o uso de padrões adicionais tais como CORBA [HER2001], UML [STY2002], etc; e o uso de novos padrões em substituição a HLA.

Alguns trabalhos têm concentrado esforços em questões de interoperabilidade da RTI. O surgimento de diferentes implementações da RTI/HLA sem interface definida impede que federados criados para uma implementação RTI possam ser utilizados em outras. Neste sentido, estudos têm proposto soluções baseadas na criação de padrão de interface RTI e a criação de pontes *RTI-to-RTI* (tradutores), entre outros [GRA2003].

Além disso, em função da necessidade de alteração do código de federados para sua integração a HLA, após a sua padronização surgiram estudos visando sua extensão para permitir o mapeamento de um SOM (federado) para FOM's diferenciadas sem a necessidade de alterações do código. Para isso, tais estudos propõem basicamente a separação de detalhes sintáticos de FOM's, o que permitiria o mapeamento de SOM's. No entanto, a necessidade de alteração de código deixa de existir apenas depois que o federado esteja preparado para pelo menos uma federação (FOM). Esta idéia é utilizada no pacote comercial denominado OMni [OMN2004].

Nas ferramentas multilinguagem, o suporte à variedade de linguagens está restrito às linguagens suportadas pela ferramenta. Embora passíveis de expansão, é necessário esforço de implementação para que um ambiente deste tipo possa suportar a inclusão de modelos desenvolvidos em uma linguagem ainda não prevista. Nas contribuições desta tese, a existência de mecanismos de interface adaptativos reduz expressivamente o esforço de adição de um novo componente descrito em uma linguagem ainda não prevista. Isto ocorre em função do zelo pela independência dos componentes e pela generalidade dos mecanismos concebidos para o suporte à execução de simulações. Embora a agregação de um novo modelo a um ambiente existente também necessite de um certo trabalho na implementação, este trabalho permanece restrito aos requisitos de interface.

Na ferramenta IPCHINOOK a interligação de um novo componente num modelo em construção é dependente do suporte oferecido pelas primitivas previstas em ACT. Esta característica limita a liberdade de reuso.

A ferramenta JavaCAD permite instanciar componentes IP de múltiplos fornecedores remotos e simulá-los em conjunto com blocos proprietários. A implementação de mecanismos de controle para impedir o fluxo de informações consideradas confidenciais entre fornecedores e usuários de componentes é uma característica vantajosa. No entanto, embora JavaCAD vise modelos heterogêneos, os componentes são dependentes de um cliente JavaCAD que deve ser implementado à luz das características do componente, ou a necessidade de adaptação recai sobre o componente. Além disso, a comunicação entre componentes distintos precisa ser negociada através de um servidor JavaCAD (seja para modelos heterogêneos ou não), centralizando o gerenciamento da simulação. O WESE também é baseado num servidor. Já no DCB não existe a figura de um servidor intermediário, pois um mesmo federado pode solicitar ou fornecer serviços. Ou seja, os federados podem se comportar tanto como servidores bem como clientes, e se comunicam diretamente, adicionando maior flexibilidade se comparado a estratégias que centralizam operações de gerenciamento em servidores.

As ferramentas de co-simulação Pia fornecem mecanismos de especificação que permitem a interconexão de nodos através de uma rede de comunicação, podendo incluir modelos distribuídos geograficamente. Os nodos podem ser interconectados com o uso de *sockets*, viabilizando a conexão com outras ferramentas de projeto tais como outros simuladores, ou compiladores, ou mesmo dispositivos. No entanto, estas ferramentas precisam ser construídas com suporte para *sockets* Pia, caso contrário uma alternativa de propósitos específicos torna-se necessária para viabilizar a conexão com ferramentas Pia. Na proposta do DCB, quaisquer restrições de comunicação que poderiam recair sobre os simuladores são tratadas pelo mecanismo de comunicação, preservando a integridade dos simuladores (independência).

Em resumo, a bibliografia apresenta variadas propostas geralmente direcionadas, em maior ou menor grau, para soluções de problemas dentro de um contexto pré-determindado ou limitado. Esta tendência é mais expressiva se observadas ferramentas comerciais (*Coware* [COW2003], *Cadence* [CAD2003], *Mentor Graphics* [SEA2003], *Synopsys* [SYN2003]), que, entre outras funcionalidades específicas, empregam também a co-simulação.

Os ambientes e ferramentas existentes tendem a não incorporar características de generalidade com o objetivo de tratar problemas inesperados, ou não previstos em um determinado domínio. Deste modo, soluções proprietárias ou baseadas em um dado conjunto de restrições podem impor um excessivo fardo em termos de reprojeto/manutenção de ferramentas para o suporte de novas características. Mais do que isso, o uso explícito de soluções não proprietárias e distribuídas, em conjunto, não tem sido visto como uma questão fundamental em abordagens existentes. O oferecimento, de modo combinado num único ambiente, de propriedades (sincronização híbrida, independência de componentes, execução distribuída) encontradas isoladamente em estudos/ambientes/ferramentas existentes justifica a relevância do DCB frente aos desafios atuais.

3 DCB: UMA ARQUITETURA DE SUPORTE À SIMULAÇÃO DISTRIBUÍDA DE MODELOS HETEROGÊNEOS

Este capítulo tem como objetivo apresentar a arquitetura do DCB (*Distributed Co-simulation Backbone*) e os princípios que conduziram sua concepção até o estágio atual. Na primeira seção é realizada uma discussão em torno das justificativas relevantes para o desenvolvimento do DCB. Em seguida, são abordadas suas funcionalidades principais, seguido da apresentação das atribuições de cada um dos módulos existentes na estrutura do DCB.

3.1 Introdução

A simulação, vista como uma técnica de apoio à validação de sistemas, contribui na detecção de problemas em fases intermediárias do processo de projeto. Quando um sistema em construção possui múltiplos e diferentes componentes a simulação, então dita heterogênea, tem sua importância valorizada na medida em que oferece meios para validar aspectos de cooperação desde fases precoces do projeto.

Dentre os maiores desafios no campo da simulação heterogênea, em especial na co-simulação (HW/SW), destaca-se a construção de mecanismos eficazes para suportar a cooperação consistente entre os componentes que integram um mesmo modelo [BOR2000]. À luz deste desafio, pesquisas na busca de técnicas, ambientes e ferramentas para co-simulação têm mostrado que um dos principais gargalos está na interface de comunicação. A grande variedade de tecnologias e sua contínua evolução dificultam a concepção de mecanismos adaptativos (ou genéricos) que possam promover a cooperação entre simuladores heterogêneos [NAN99].

Os componentes de um modelo de simulação podem ser sub-modelos, componentes IP, dispositivos físicos (HW), entre outros, que mantenham comunicação com seu exterior. Neste texto, o termo federado é utilizado para designar os componentes que, integrados em um único modelo, formam uma federação. O termo componente é utilizado no texto como sinônimo de federado.

Os desafios diante da construção de ambientes de co-simulação têm motivado pesquisas em busca de abordagens mais flexíveis [STR98]. No escopo do estado da arte em simulação heterogênea, alguns trabalhos existentes propõem o uso de interfaces padrão para viabilizar a comunicação entre componentes incompatíveis. Esta alternativa possui dois inconvenientes principais, a necessidade de modificações no componente e os limites da abrangência do padrão definido para a interface.

Inicialmente, o uso desta política pode interferir na construção de componentes novos e provocar a modificação de componentes existentes para que possam ser reutilizados em modelos para os quais não foram previstos. Neste caso, as chances de reuso de componentes na forma original sofre redução devido a dois fatores principais: o retrabalho para sua modificação pode ser demasiadamente grande e o reuso pode ser inviabilizado se consideradas questões de segurança e propriedade intelectual.

Por outro lado, o conjunto de definições de uma interface padrão pode limitar a abrangência de situações tratáveis (não suportando as características de interface de um componente com características distintas ou não previstas). Neste caso, apenas a modificação de um componente para adaptá-lo à interface pode não ser suficiente para o seu reuso. Estes fatores justificam tendências ao não uso de interfaces rígidas e a preferência às pesquisas na busca de mecanismos que sejam capazes de reconhecer múltiplas (embora finitas) interfaces.

Contudo, mesmo capacitando um ambiente de simulação para que reconheça N interfaces, as restrições quanto ao reuso ainda se mantêm, pois um federado cuja interface não esteja prevista no ambiente também exigirá modificações para que possa ser integrado. Esta restrição pode ser visualizada em ambientes que suportam um conjunto finito de linguagens, onde o federado precisa se manter restrito a um número também finito de possibilidades de interface. Neste caso, a inclusão de um outro formato para interface significa a realização de modificações no ambiente/ferramenta, o que pode exigir esforço considerável, ou no componente para que seja adaptado a uma das interfaces suportadas.

Embora problemas relacionados à interface de comunicação exijam esforço adicional para incorporação de um novo componente, ou imponham limites ao reuso de tecnologias/linguagens por um determinado ambiente, os desafios em simulação heterogênea não se restringem a problemas de interface. Eles são extensivos, por exemplo, para o controle da ocorrência de eventos que precisam considerar aspectos de sincronização e tratamento de dados. Nestes casos, para que um novo componente possa ser utilizado num modelo, mesmo que o ambiente de simulação reconheça sua interface, este componente pode não ser compatível com as políticas de sincronização e tratamento de dados implementada pelo ambiente. O mesmo pode ocorrer com o protocolo que rege a cooperação.

Problemas deste tipo causam tanto transtornos quanto problemas de interface de comunicação, pois podem exigir alterações no componente, ou mesmo impor regras na construção de novos componentes. Esta característica também reduz as perspectivas de reuso, principalmente no caso de componentes IP.

Em resumo, geralmente as abordagens e ferramentas existentes são adequadas para solucionar um conjunto pré-definido de problemas. Contudo, freqüentemente situações não previstas precisam ser tratadas. Neste caso, soluções que são proprietárias, ou baseadas em um dado conjunto de restrições, podem impor um excessivo desgaste em termos de reprojeto [PAG2000].

Tais dificuldades tornam-se críticas no campo da simulação heterogênea distribuída, principalmente quanto às políticas de sincronização. No entanto, a possibilidade de distribuição física dos federados é desejável, o que motiva o desenvolvimento de estudos nesta área [CHE2001] [SCH2003].

A bibliografia apresenta contribuições em simulação distribuída, algumas delas no suporte a modelos heterogêneos, como por exemplo ipChinook e JavaCAD, entre

outras discutidas no Capítulo 2. Atualmente, a arquitetura HLA incorpora características que a tornam uma das alternativas mais flexíveis no gerenciamento de simulações distribuídas. Contudo, embora HLA possua recursos de suporte a modelos heterogêneos, teve suas propriedades principalmente definidas em função da simulação interativa distribuída com propósitos de treinamento militar [FUL96].

Além disso, HLA não prevê a preservação da integridade do código dos federados. No entanto, recentemente têm surgido trabalhos que buscam alternativas, no contexto de HLA, com o objetivo de evitar a modificação no código dos federados [TOL2002]. Esta preocupação, em conjunto com a independência dos federados em termos de interface [ROW97], são os requisitos fundamentais utilizados já na definição da arquitetura do DCB, proposta nesta tese.

Assumindo HLA como a referência mais flexível no suporte à simulação distribuída, baseado no estudo de trabalhos correlatos, a concepção do DCB teve sua inspiração inicial em HLA. Um dos fatores motivadores para o uso de HLA foi a sua padronização em setembro de 2000 (IEEE Standard 1516/1516.1/1516.2) [1516]. Embora reconhecida como padrão pela IEEE, recentemente o DMSO [DMS2004] (com participação da SISO [SIS2004]) emitiu um documento manifestando que o padrão IEEE 1516 não interpreta corretamente os princípios de HLA e, assim, descrevendo correções de interpretação [DOD2003].

Afora as discussões entre IEEE e DMSO/SISO em relação à interpretação das especificações HLA, é visível a expansão do seu uso em pesquisas e aplicações finais que envolvam simulação distribuída.

A primeira publicação apresentando a arquitetura do DCB [MEL2001a] o cita como sendo "baseado" em HLA. Contudo, no andamento da tese, as políticas implementadas pela arquitetura do DCB foram perdendo a similaridade com as definições de HLA. Deste modo, fez-se mais adequado apresentar o DCB como uma proposta "inspirada" a partir das características do padrão HLA.

Como justificativa desta afirmação, é relevante levar em consideração alguns aspectos:

1. O DCB propõe, desde sua concepção, a preservação máxima do código do federado. Contudo, em HLA, existe um conjunto de regras [KUH2000] que precisa ser respeitado pelo federado para que ele possa ser agregado a uma federação. Este requisito, que exige modificações no código do federado, não é aceito na proposta do DCB;
2. HLA também possui regras que precisam ser respeitadas por federações. No caso do DCB, algumas observações são feitas apenas no escopo do trabalho de configuração de uma federação, não de sua execução;
3. Em HLA, o federado precisa reconhecer as primitivas RTI disponíveis, bem como possuir a consciência da existência dos federados com quem deseja manter cooperação. Como o DCB propõe independência dos federados, este requisito é igualmente indesejável;
4. Em HLA, o termo "embaixador" é utilizado para designar as requisições existentes dentro do código dos federados para instanciar primitivas RTI. No DCB o termo embaixador é utilizado para designar os módulos internos da arquitetura do DCB, localizados fora do código dos

federados, que executam as funções de gerenciamento necessárias. Estas funções, invisíveis aos federados, são descritas no decorrer deste capítulo;

5. O DCB possui um módulo de comunicação que se assemelha à RTI HLA. Este é o item de maior semelhança entre ambos. Contudo, no DCB, atividades de gerenciamento da execução de federações (simulação) são compartilhadas entre os embaixadores e o núcleo do DCB (NDCB). Operações de comunicação com nodos remotos são exclusivas do NDCB. Em HLA, a RTI centraliza todas as primitivas de gerenciamento necessárias à execução de uma simulação.

Observa-se nas considerações acima que, embora inspirado em HLA e com os objetivos gerais semelhantes, o DCB tem sua concepção motivada pelas necessidades de validação de sistemas heterogêneos (essencialmente co-simulação) e pela busca de independência de federados. Estes fatores conduziram o trabalho de concepção do DCB visando uma arquitetura que busca reduzir expressivamente a imposição de regras/restrições sobre federados, sejam quais forem suas finalidades.

A hipótese de fazer um trabalho com os mesmos objetivos, contudo mantendo-se fiel ao padrão HLA, fatalmente implicaria em: extinção das regras dos federados, criação de ao menos uma camada interna entre federado e RTI, alteração das regras de federação, alterações estruturais e conceituais nas primitivas RTI, entre outros. Tais alterações iriam desfigurar os conceitos essenciais do padrão HLA, motivo pelo qual a alternativa de basear o DCB no padrão HLA foi abandonada.

Contudo, afora as restrições HLA para construção e execução de modelos de simulação heterogênea, alguns aspectos justificam o uso das definições conceituais desta arquitetura como base para ambientes de co-simulação. São eles:

- É um padrão, o que o torna atrativo como referencial para o desenvolvimento de simuladores;
- É mais flexível que abordagens atuais, permitindo uma definição mais fácil de federações em particular; e
- Prevê a integração de simuladores existentes, embora impondo regras e a necessidade de modificações.

Deste modo, é válido dizer que o DCB foi inicialmente ‘inspirado’ em HLA, mantendo inclusive alguns ítens similares. Contudo, é fortemente voltado para independência de federados, aspecto não previsto nas especificações do padrão HLA.

O propósito principal deste capítulo é a apresentação da arquitetura do DCB cujas funcionalidades abrangem o suporte à sincronização e gerenciamento de dados, de interface e de propriedades sobre atributos de entrada e saída de federados, ambos considerando distribuição física das partes que compõem um modelo. Para isso, propõe abstrair dos federados a necessidade de interação com os serviços de simulação. Deste modo, o comportamento interno do componente (código) é preservado, o que adiciona flexibilidade na exploração do reuso de componentes na construção e execução de modelos de co-simulação distribuídos.

Embora a ênfase deste trabalho seja o uso da co-simulação na validação de sistemas embarcados, o DCB pode incorporar características diferenciadas para outros domínios (ou espécies/tipos) de modelos. Tais diferenças, por exemplo, podem interferir em requisitos de desempenho, de sincronização, ou de restrições temporais. Por exemplo, um modelo de co-simulação de sistemas eletrônicos, um ambiente com *components-in-the-loop* e um ambiente de treinamento (*human-in-the-loop*) possuem características sensivelmente diferenciadas que podem exigir implementações também diferenciadas da arquitetura do DCB, ou mesmo das estratégias de modelagem/configuração de federações.

Nas seções a seguir, inicialmente é feita uma abordagem geral da arquitetura do DCB. Em seguida são apresentadas as funcionalidades internas de gerenciamento, sem contudo atribuí-las a módulos específicos do DCB. Após esta apresentação, são identificados então os módulos do DCB e seus objetivos de acordo com as funcionalidades definidas. Por fim, a última seção discute as contribuições do DCB.

3.2 Visão geral da arquitetura do DCB

O DCB é composto por quatro módulos principais: o Embaixador do DCB (EDCB), o Embaixador do Federado (EF), o Núcleo do DCB (NDCB) e o *gateway*. O *gateway* tem como principal tarefa o tratamento das interfaces dos componentes. Os demais tratam da sincronização, gerenciamento de dados e cooperação. A Figura 3.1 apresenta a arquitetura do DCB.

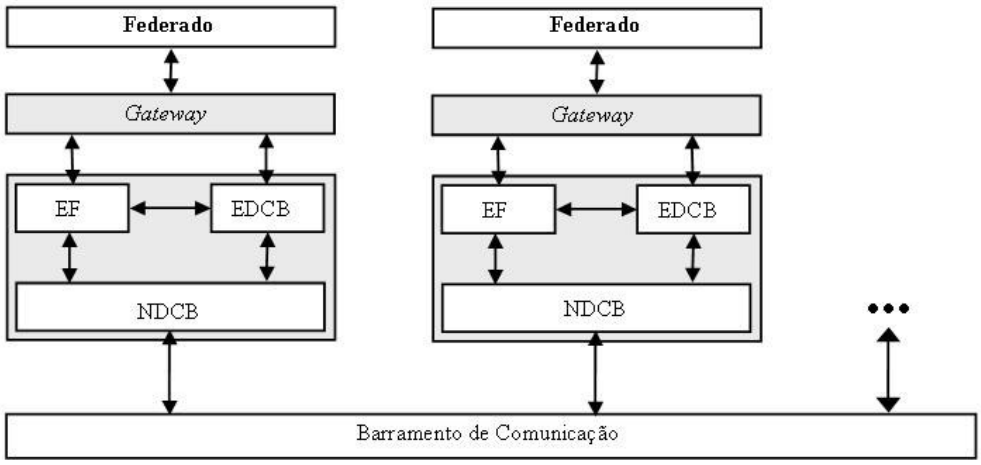

Figura 3.1: Arquitetura do DCB

O EF tem como propósito geral o gerenciamento de mensagens recebidas de outros federados, sejam eles remotos ou locais. É ele quem realiza a decodificação de pacotes recebidos e participa das atividades de gerenciamento do tempo de simulação.

O EDCB tem como propósito principal o gerenciamento de mensagens emitidas pelo federado que ele representa. Em conjunto com o EF, também mantém gerenciamento do tempo virtual local (LVT-*Local Virtual Time*) e do modo de sincronização utilizado por cada federado para cooperar com a federação. A

sincronização pode ser híbrida, ou seja, federados síncronos, por exemplo, podem cooperar com federados que não utilizam tempo.

O NDCB, além de manter os serviços de comunicação por troca de mensagens, mantém um valor único em todos os nodos para o tempo virtual global (GVT-*Global Virtual Time*). O GVT é atualizado a partir da leitura do tempo de evento de todas as mensagens trocadas entre federados.

Devido à heterogeneidade dos federados que podem cooperar em um ambiente de co-simulação, a implementação de uma interface de comunicação torna-se uma tarefa complexa. Com o objetivo de tratar esta complexidade mantendo bons níveis de flexibilidade na agregação de novos federados, o DCB utiliza o conceito de *gateway* [STR98]. *Gateways* incorporam a capacidade de traduzir dados de um formato origem para um formato destino, de acordo com exigências sobre dados enviados através do DCB. O *gateway* é o módulo de ligação (interface) entre o federado e os mecanismos de suporte à simulação (EF, EDCB, NDCB).

Para cada federado que é integrado ao ambiente de co-simulação, é necessário que um embaixador do federado e um embaixador do DCB sejam inicializados no mesmo nodo em que se encontra o federado. Assim, se uma federação inclui ‘n’ federados, mesmo que eles sejam homogêneos, ou seja, implementados com o uso da mesma tecnologia/linguagem, como por exemplo VHDL [VHD2000], ‘2n’ diferentes embaixadores serão necessários para integrar ‘n’ federados a uma federação.

De acordo com a arquitetura do DCB, os federados não precisam manter informações sobre a federação. Eles apenas reconhecem a existência do *gateway* com quem mantêm troca de dados através de atributos de interface. As atividades de gerenciamento da cooperação entre federados ocorrem além do *gateway*. Estas atividades não são visíveis aos federados (independência dos federados). Por este motivo os federados não precisam invocar primitivas de comunicação do DCB, como acontece na RTI/HLA, para envio de dados (entre outros serviços). O federado apenas disponibiliza dados de saída ou recebe dados a ele enviados utilizando os atributos de interface gerenciados pelo *gateway*. Estes atributos precisam ser pré-definidos no momento da construção da federação. O EDCB mantém gerenciamento sobre mensagens a serem enviadas, e o EF sobre mensagens recebidas. As operações de comunicação são mantidas pelo NDCB.

No gerenciamento da cooperação entre federados, embora a comunicação ocorra no formato *client*/*server*, servidores e clientes não são fixos. Ou seja, cada federado pode se comportar tanto como cliente quanto como servidor de acordo com o estado em que se encontra e da direção dos dados em trânsito. Um federado pode estar prestando um serviço para outro federado, ou pode estar requisitando um serviço. Outro ponto relevante é o de que o código do DCB e dos embaixadores é o mesmo em todos os nodos envolvidos em uma federação. No tratamento de interfaces heterogêneas a responsabilidade pela tradução de dados é restrita ao *gateway*.

Embora a existência do *gateway* como um módulo de interface para o federado possa implicar em algum esforço de implementação quando um novo federado é adicionado, ele evita a implementação de modificações em outras partes da federação, por exemplo na infra-estrutura de comunicação e sincronização (núcleo do DCB), ou mesmo em detalhes internos do federado. Tais modificações implicariam em maiores custos se comparadas à construção de *gateways*.

Os mecanismos de suporte à comunicação são equivalentes tanto para a troca de mensagens entre federados locais (que estão no mesmo nodo) quanto para federados remotos. A justificativa para o não uso de mecanismos diferenciados para troca de mensagens entre federados remotos e locais está baseada em dois aspectos principais: a homogeneidade dos instrumentos de controle do tempo de simulação e a possibilidade de trabalhar de forma colaborativa sobre um mesmo projeto. Embora, no protótipo atual, a comunicação com federados locais ocorra através de chamada de métodos (totalmente interno ao DCB) ao invés do uso de *sockets*, usados na comunicação com federados remotos, o tratamento de mensagens se mantém idêntico para ambos.

O trabalho colaborativo sobre um mesmo projeto também é beneficiado pela não diferenciação na forma de comunicação entre federados locais e remotos. Por exemplo, um projeto de co-simulação pode ser desenvolvido em conjunto por dois projetistas, um deles localizado no nodo A e o outro no nodo B. Depois de pronto o modelo, sua execução é iniciada a partir de um nodo C numa primeira replicação e a partir de um nodo D numa segunda replicação. Neste exemplo, observa-se que os nodos considerados locais na replicação disparada a partir de C passam a ser considerados remotos na replicação disparada a partir de D. O mesmo pode ser observado nas demais combinações entre os nodos A, B, C e D, tanto para projeto bem como para o disparo de execuções.

Também é desejável que ambientes de simulação baseados na Web, e que permitem o compartilhamento e o reuso de federados, além de otimizar o desenvolvimento de sistemas, possam oferecer o recurso de substituição de federados em tempo de execução. Nestas condições, o projetista (ou usuário) pode alterar de forma seletiva diferentes partes de uma federação, agregando flexibilidade na construção e execução de modelos. Diz-se de forma seletiva [RAO2000b] [HIN97c], pois a troca pode ser efetuada sobre um componente em particular ou sobre um conjunto de componentes de um modelo. A troca seletiva oferece um mecanismo mais eficaz ao projetista para otimizar a relação entre *overhead* de comunicação e necessidade de detalhamento dos federados (níveis de abstração). Além disso, a estrutura do DCB torna a inclusão/exclusão de federados natural, ou seja, sem a necessidade de mecanismos de apoio. No ambiente WESE, por exemplo, esta tarefa exigiu a construção de um módulo de apoio (biblioteca) denominado DCS (discutido no Capítulo 2).

Embora o protótipo atual do DCB já permita que ocorra a substituição de federados em tempo de execução, a situação onde um federado origem envia mensagem a um federado destino que não está executando ainda requer mais esforço de programação na parte da implementação que controla o uso de *sockets* para gerenciamento das conexões ativas. Nas situações em que um federado a ser substituído não recebe nenhuma mensagem durante o tempo que estiver inativo, a substituição é suportada pelo protótipo.

3.2.1 Identificação de federados e federações

A identificação de um federado em particular permite personalizar cada federado de uma co-simulação. Mesmo que existam dois federados idênticos (replicados) em uma mesma federação, eles serão tratados de forma independente pelo DCB. A identificação de federados pode ser atribuída de forma automática pelo DCB, ou realizada pelo projetista. O identificador do federado precisa ser adicionado a todas as mensagens trocadas entre federados através do DCB.

Da mesma forma, um identificador de federação é necessário nas situações em que duas federações distintas cooperam. Por exemplo, a conexão entre diferentes federações ocorre quando um atributo de saída de um federado pertencente a uma federação X é conectado a um atributo de entrada pertencente a um federado de uma federação Y.

Neste caso, existe o risco de dois federados pertencentes a federações distintas possuírem o mesmo identificador, o que resulta numa situação que pode ser interpretada de modo equivocado pelo DCB. Isto justifica a existência de uma identificação também para federações. Assim como para federados, a identificação da federação também precisa ser adicionada a todas as mensagens trocadas entre federados.

3.3 Gerenciamento de dados e de tempo e aspectos de comunicação no DCB

O gerenciamento de dados e de tempo realizado pelo DCB é inspirado no padrão HLA (IEEE 1516). Contudo, a arquitetura do DCB permite abstrair dos federados as operações de gerenciamento, preservando sua integridade. Esta propriedade do DCB agrega generalidade e flexibilidade para a construção e execução de modelos heterogêneos.

Basicamente, em comparação com HLA, o DCB busca reduzir a imposição de regras sobre federados e federação. Em HLA, serviços de gerenciamento e comunicação disponíveis na RTI precisam ser explicitamente instanciados pelo federado, o que exige o acesso aos detalhes de sua implementação. Este fato vai de encontro à idéia de preservação de integridade de federados, relevante especialmente para questões de reuso.

Na especificação do gerenciamento de dados, a arquitetura DCB define uma política de exclusão mútua para o uso de atributos de interface que são compartilhados entre os federados. Deste modo, somente um federado, num instante de tempo, possui propriedade sobre um determinado atributo e pode alterar seu valor. Além disso, os mecanismos do DCB voltados ao gerenciamento de propriedade de atributos são totalmente independentes do federado. Este fator contribui grandemente com a preservação de sua integridade.

O mesmo é verdade para o gerenciamento de tempo. O DCB é auto-suficiente para o gerenciamento de tempo, contudo respeitando as características dos federados quanto ao controle de tempo por eles mantido. Por exemplo, se um federado não utiliza tempo, o DCB permite que este federado seja adicionado a uma federação híbrida (com federados síncronos e assíncronos). Para isso, o DCB executa ações de controle dos eventos externos desses federados de modo que tais eventos se mantenham consistentes na federação em relação ao relógio global de simulação.

A seguir são apresentados detalhes do DCB sobre retorno a estados seguros para modelos assíncronos, gerenciamento de propriedade, gerenciamento de tempo e aspectos de comunicação. A partir da Seção 3.4, são apresentadas as atribuições de cada módulo do DCB.

3.3.1 Gerenciamento de propriedade

A parte do DCB que trata do gerenciamento de propriedade tem como tarefa principal manter uma política de exclusão mútua sobre o uso dos atributos de interface pelos federados. Ou seja, utilizando como referência um conjunto de condições pré estabelecidas pelo projetista (de modelos de co-simulação), o gerenciamento de propriedade do DCB faz com que um único federado tenha acesso a um mesmo atributo destino num determinado instante de tempo.

Para isso, o DCB mantém registro da identificação dos atributos e do respectivo federado proprietário. Estas informações são replicadas em todos os nodos que possuem federados ativos de uma mesma federação. A replicação destas informações é importante para facilitar e agilizar o trabalho de atribuição e cancelamento de propriedade sobre atributos.

O DCB realiza gerenciamento de propriedade apenas sobre os atributos de entrada, pois é inerente o uso dos atributos de saída apenas pelos federados que os possuem em sua interface. Além disso, a propriedade sobre atributos pode ser dinâmica (um atributo pode pertencer a n federados durante uma simulação) ou estática (um atributo pertence a um único federado durante toda a simulação).

Quando houver ligação entre atributos de federados que estão alocados em diferentes federações, as informações de propriedade dos atributos existentes em ambas as federações precisam estar replicadas em todos os nodos dessas federações. Deste modo, a existência de múltiplas federações num mesmo modelo de co-simulação não requer alterações nas estratégias de gerenciamento de propriedade, se comparadas às estratégias utilizadas para uma única federação.

A arquitetura do DCB permite que as políticas de gerenciamento de propriedade sejam mantidas de modo totalmente independente dos federados. Ou seja, quando um federado utiliza um atributo de saída para envio de mensagem a um federado destino, o DCB inicialmente armazena esta mensagem em uma fila de saída mantida pelo EDCB. A partir daí o federado considera a mensagem enviada. Contudo, antes do envio de fato da mensagem, o DCB verifica se o federado possui propriedade sobre o atributo destino. Se positivo, envia a mensagem, retirando-a da fila de saída. Caso contrário, requisita a propriedade do atributo destino de acordo com um conjunto de regras (discutidas a seguir). Ao adquirir a propriedade sobre o atributo destino, a mensagem é então enviada.

Para que operações de gerenciamento de propriedade possam ser executadas pelo DCB, um conjunto de regras para cada atributo precisa ser fornecido ao EDCB. O fornecimento de regras acontece no mesmo momento em que um projetista cadastra os atributos dos federados participantes da federação (configuração). A princípio, estas regras podem ser enquadradas em três categorias: ou por uma relação comparativa entre os LVTs dos federados; ou por uma relação comparativa entre o LVT do federado que solicita uma propriedade e o GVT; ou ainda pela propriedade sobre um conjunto de atributos que deve estar previamente estabelecida.

Para exemplificar as categorias em que podem se enquadrar as regras para gerenciamento de propriedade, são considerados os federados A, B e C, interconectados de acordo com a Figura 3.2. Cada federado possui um atributo de saída de dados utilizando o índice ‘O’ (AO) e um atributo de entrada dado usando o índice ‘I’ (BI, CI). A configuração dos atributos entre estes federados é a seguinte:

- AO ⇢ BI (AO requer propriedade de BI);
- AO ⟶ CI (AO possui propriedade de CI).

Os seguintes casos ilustram três possibilidades de regras que podem ser utilizadas como requisito para o atendimento a pedidos de propriedades sobre atributos de outros federados. Nestes casos, o federado A deseja enviar um valor de AO para o atributo BI do federado B, necessitando requisitar sua propriedade.

- Se LVT(A) < LVT (B), então a requisição de A para obter a propriedade de BI é atendida (Figura 3.2 a);
- Se LVT(B) < GVT (considerando que o valor de um LVT pode ser igual ao GVT), então a requisição de A para obter a propriedade de BI é atendida (Figura 3.2 b);

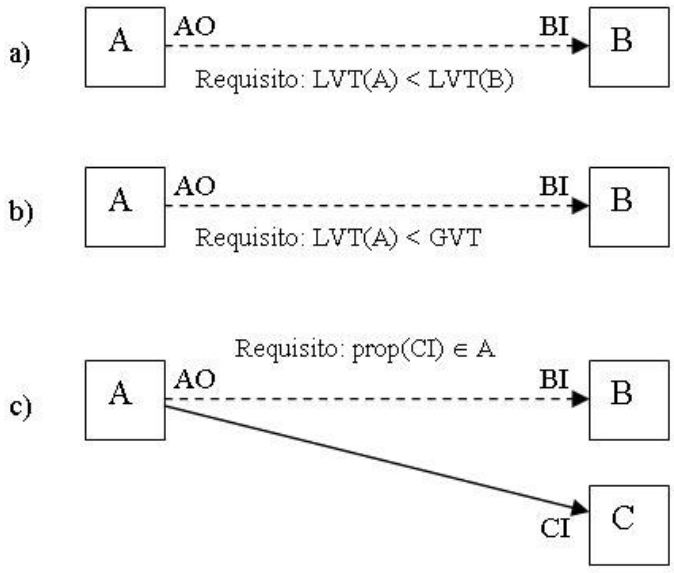

Figura 3.2: Exemplo do uso de regras para atribuição de propriedade sobre atributos.

- Se A possui propriedade de CI, então a requisição de A para obter a propriedade de BI é atendida (Figura 3.2 c). Caso contrário, uma requisição para atribuir a propriedade de CI para A é automaticamente gerada. Quando confirmada, a requisição original de A por BI é então atendida. Nesta situação, um federado deve possuir a propriedade de dois atributos de entrada simultaneamente para que possa utilizá-los. Neste exemplo hipotético, A deve possuir CI para que possa possuir BI. Esta regra é útil quando um federado precisa enviar valores, de modo contíguo, para dois atributos de entrada distintos.

Nas ligações entre atributos sujeitas a uma ou mais regras de propriedade, o DCB é responsável pelo atendimento das regras antes de autorizar o uso do atributo destino perante uma solicitação do federado origem. As regras para atribuição de propriedade são fornecidas pelo projetista em conjunto com o trabalho de configuração de uma federação.

Como os requisitos para envio e recebimento de dados entre federados são dependentes dos propósitos do modelo em construção, não há razões para exigir o estabelecimento de regras para todas as ligações entre atributos mantidas em uma federação. Por isso, a existência de regras para alocação de propriedade sobre atributos é vista como um instrumento adicional que pode ser utilizado para resolver situações particulares ou atípicas na cooperação entre federados. A única regra padrão é a de que um atributo de entrada não seja alocado a dois outros atributos de saída simultaneamente. Deste modo, ao receber uma solicitação de propriedade sobre um atributo de destino, o DCB verifica a existência ou não de regras para aquela ligação. Se houverem, elas são atendidas, caso contrário, a propriedade é concedida imediatamente se o atributo em questão não estiver alocado a um outro atributo de saída. Se estiver, a propriedade do atributo é atribuída ao solicitante assim que ele for liberado pelo federado que a detém.

Uma vez estabelecida a configuração dos atributos de interface e das regras de propriedade, o gerenciamento de propriedade mantido pelo DCB ocorre de acordo com a seqüência de passos a seguir:

1. Federado ‘A’ altera valor do atributo de saída ‘AO’ configurado para enviar dados ao atributo de entrada ‘BI’ do federado ‘B’;
2. Se o federado ‘A’ (gerenciado através de seu EDCB) não possui a propriedade de ‘BI’, é gerado um evento interno que requisita a propriedade de ‘BI’ para ‘A’. Caso ‘A’ possua a propriedade de ‘BI’, passa para a fase 4;
3. O DCB faz a leitura das regras de propriedade do atributo ‘BI’, registradas no EDCB do federado ‘A’, e executa as ações necessárias para atender às regras (se houverem). Em seguida, se ‘BI’ não estiver alocado a nenhum outro federado, sua propriedade é atribuída para ‘A’. Se outro federado possuir ‘BI’, este atributo deve ser liberado antes que ‘A’ receba o direito de utilizá-lo (exclusão mútua no uso de atributos de entrada);
4. A mensagem com o novo valor de ‘AO’ é entregue em ‘BI’;
5. O atributo ‘BI’ é então disponibilizado pelo EDCB, para que outro federado possa requisitar sua propriedade.

A política de gerenciamento de propriedade proposta no DCB permite uma independência total do federado, o que contribui com os propósitos gerais de preservação da integridade dos componentes que participam de uma simulação. Esta característica é importante para facilitar e agilizar o reuso de componentes, inclusive IP’s.

3.3.2 Gerenciamento de tempo

As funcionalidades do DCB que executam atividades de sincronização têm como tarefa básica manter o controle sobre o tempo de evento das mensagens trocadas entre federados de acordo com o GVT (*Global Virtual Time*). O tempo de evento enviado em uma mensagem indica o tempo em que um evento deve ocorrer no federado destino.

Não faz parte dos objetivos do DCB apresentar contribuições no modo como é realizado o trabalho de sincronização entre federados. A bibliografia apresenta uma análise bastante ampla sobre evolução conservadora e otimista do tempo em simulação distribuída [FUJ2001]. Contudo, embora o DCB utilize os conceitos clássicos de sincronização, as políticas adotadas foram adaptadas para estarem em consonância com as características de flexibilidade, generalidade e independência de federados proposta pela arquitetura do DCB.

Por isso, o DCB busca 'abstrair' dos federados a necessidade de tomar qualquer iniciativa para requisitar serviços do DCB buscando cooperação com demais federados da federação de que participam. Assim, cabe aos federados apenas manter gerenciamento dos atributos de sua interface. O que acontece além dos seus atributos é então ignorado pelos federados.

O DCB suporta sincronização híbrida, permitindo que cada federado avance o seu tempo local no modo assíncrono ou no modo síncrono [DAH97], assim como a cooperação com federados que não utilizam tempo explicitamente no seu modelo. Para referenciar este tipo de federados, o termo '*notime*' é utilizado.

Em ambientes distribuídos, o modo assíncrono geralmente apresenta menos *overhead* na medida em que não limita ao GVT a evolução dos LVT's. No entanto, para que um federado execute no modo assíncrono, ele precisa suportar operações de retorno para estados seguros perante a ocorrência de violações de LCC [ELN99]. Para isso, um federado pode possuir uma política de retorno internamente implementada, ou apenas reconhecer anti-mensagens [ELN99] para retroceder no tempo. Tendo em vista que uma federação suportada pelo DCB é configurada de acordo com as características de cada federado, o DCB permite que federados de ambos os tipos (com algoritmo de retorno interno, ou apenas com reconhecimento de anti-mensagens) possam ser agregados em uma mesma federação.

Visando prover recursos para a execução de retornos, o DCB permite o armazenamento de mensagens enviadas e recebidas de cada federado. Este serviço é realizado pelos embaixadores, sem o envolvimento dos federados. Com esta política, basta que os federados assíncronos reconheçam anti-mensagens para que retrocessos no tempo possam ser controlados pelo DCB de modo consistente.

No modo síncrono, o DCB utiliza o serviço de gerenciamento de propriedade, em conjunto com o LVT dos federados e do GVT para garantir que somente eventos seguros sejam executados. Por exemplo, supondo a existência de dois federados denominados A e B, e A envia uma mensagem que modifica o valor de um atributo X de B. O DCB deve conceder a propriedade de X para A permitindo que A modifique o seu valor. Contudo, por exemplo, esta atribuição de propriedade somente pode ser efetivada se o LVT de A for maior ou igual ao LVT de B.

Com o objetivo de otimizar o desempenho dos federados que executam no modo síncrono, está prevista para o DCB a implementação do mecanismo de predição de

comportamento (*lookahead*) [FUJ90]. A 'predição de comportamento' é um período de tempo L no qual há garantias de que nenhuma solicitação para a execução de eventos será feita, ou seja, é um período seguro. Seu valor pode ser determinado de forma estática ou dinâmica. Na forma estática, o projetista fornece um valor para L que é utilizado durante toda a simulação. Na dinâmica, o valor de L é alterado em tempo de execução de acordo com a quantidade de tempo entre a ocorrência de eventos. Esta quantidade de tempo precisa ser constantemente calculada. Se L é o valor da 'predição', então a atualização de um atributo X de um federado B, por outro federado A, é considerada segura se (LVT(A)+L) ≥ (LVT(B)).

Por fim, o uso de diferentes bases de tempo numa mesma federação não é previsto no DCB. Ou seja, o tempo é gerido como uma unidade equivalente para todos os federados. Assim, uma federação deve ser construída pelo projetista sobre uma única base de tempo (p.ex. *ms*). Se um federado mantém o controle do seu tempo local sobre uma base diferente, este tempo deve ser convertido (pelo *gateway*) para a base utilizada pela federação sempre que o federado fornecer seu tempo local ao DCB.

3.3.2.1 Manutenção do GVT

Mantido pelo DCB com base no tempo local de simulação dos federados (LVTs), o valor do GVT possui duas interpretações distintas: o GVT síncrono utilizado no controle do tempo de federados conservadores; e o GVT assíncrono, utilizado para federados otimistas.

Na bibliografia, o GVT síncrono é tido como igual ao maior LVT dentre os LVTs dos federados. No entanto, o DCB utiliza o conceito de *lookahead* no tratamento do GVT síncrono. Deste modo, seu valor é dado por: menor LVT + *lookahead*.

O GVT assíncrono é dado pelo menor LVT dentre os LVTs dos federados. Ele é utilizado, por exemplo, como indicador para descarte de estados seguros salvos com propósitos de retorno perante ocorrência de violações de LCC.

A possibilidade de cooperação entre modelos síncronos e assíncronos numa mesma federação exige que o DCB reconheça o modo de execução de cada federado (através de atributos) e mantenha atualizado o GVT adequado para ambos os casos (síncrono/assíncrono).

Para que o GVT seja mantido atualizado em todas as replicações dos diferentes nodos, o DCB faz com que cada atualização de LVT executada por qualquer federado gere automaticamente uma atualização do valor do GVT. Contudo, esta atualização ocorre somente se o GVT sofrer alteração perante a chegada de um novo LVT. O mesmo procedimento é utilizado para gerenciamento de ambos, GVT síncrono e assíncrono.

Já o LVT é controlado de modo explícito pelos federados que executam nos modos síncrono e assíncrono. O tempo local destes federados é 'respeitado' pelo DCB ao mesmo tempo em que o DCB controla sua evolução de acordo com o tipo de sincronização utilizado pelo federado respectivo.

Por exemplo, se o federado é síncrono, o DCB não permite que o LVT do federado ultrapasse o GVT síncrono somado ao *lookahead* . Ao mesmo tempo, não permite que o federado faça requisições para execução de eventos externos com tempo de evento menor ou igual ao GVT+*lookahead.*

Já os federados *notime* desconhecem a variável tempo na comunicação com o restante da federação. Apesar disso, para que a cooperação se mantenha correta, os EDCBs que representam federados *notime* geram automaticamente um tempo consistente em relação ao GVT+*lookahead* para garantir a ordem global de execução dos eventos externos solicitados por federados deste tipo (através do envio de mensagens). No entanto, se mais de uma mensagem é enviada a partir de um mesmo federado *notime* sem que ocorra avanço do tempo global, o tempo de evento (*timestamp* [FUJ90]) das mensagens é incrementado automaticamente e utilizado com propósito de ordenação na entrega das mensagens no destino. Ou seja, assim que mensagens provenientes de federados *notime* são inseridas na lista de entrada mantida pelo EF do destino, elas são entregues ao federado ordenadas pelo seu *timestamp* sem a necessidade de sincronização com o tempo global. Para isso estas mensagens são identificadas na origem e manipuladas de modo diferenciado pelos EFs no destino.

3.3.2.2 Modos de execução

Em relação ao modo de execução perante o tempo, a comunicação entre federados pode ocorrer de acordo com quatro possibilidades:

1. Síncrono para síncrono: se o tempo de evento da mensagem (verificado pelo DCB) for menor que o GVT síncrono, uma resposta é devolvida ao federado origem com mensagem de erro de violação de LCC. Caso contrário, a comunicação é autorizada pelo DCB;
2. Síncrono para assíncrono: mesmo quando federados destino são assíncronos, se o tempo de evento da mensagem (verificado pelo DCB) for menor que o GVT síncrono, uma resposta é devolvida ao federado origem com mensagem de erro de violação de LCC. Caso contrário, a comunicação é autorizada pelo DCB. Se o tempo de evento da mensagem for menor que o LVT do federado assíncrono destino, então um algoritmo de retorno é disparado (esta operação é realizada pelo EDCB através do disparo de anti-mensagens que são interpretadas e transmitidas pelo DCB, ou pode ser implementada pelo próprio federado);
3. Assíncrono para síncrono: se o tempo de evento da mensagem (verificado pelo DCB) for menor que o GVT síncrono, uma resposta é devolvida ao federado origem com mensagem de erro de violação de LCC. Neste caso, como um federado assíncrono não prevê o uso do GVT síncrono, o EF e o EDCB precisam interferir na geração do *timestamp* da mensagem verificando o *timestamp* em relação ao GVT síncrono. Caso contrário, a comunicação é autorizada pelo DCB;
4. Assíncrono para assíncrono: Se o *timestamp* da mensagem for menor que o LVT do federado assíncrono destino então um algoritmo de retorno é disparado (quem faz isso é o EDCB através do disparo de anti-mensagens que devem ser interpretadas e transmitidas pelo DCB, ou o próprio federado caso ele implemente ações de retorno a estados seguros);

Além destas quatro possibilidades, federados do tipo '*notime*' também podem participar de uma federação híbrida. Neste caso, as possibilidades para cooperação entre

federados de diferentes tipos aumentam para 9. A Figura 3.3 apresenta as possíveis combinações.

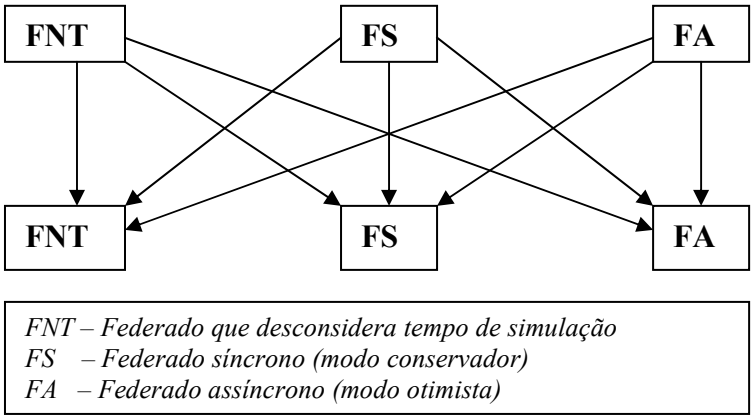

Figura 3.3: Relação entre federados em função da sincronização

Pode-se observar que federados síncronos, mesmo quando enviam mensagens para federados assíncronos, respeitam o limite mínimo para o tempo de evento dado pelo GVT síncrono. Isto é necessário para proteger os federados síncronos de possíveis situações que exijam retorno. Mesmo assim, um federado síncrono pode receber uma anti-mensagem de um federado assíncrono cujo *timestamp* é maior que o GVT síncrono + *lookahead*. Neste caso, duas situações distintas podem acontecer:

1. O federado síncrono ainda não executou o evento recebido pela mensagem correspondente à anti-mensagem. Neste caso, a mensagem original é simplesmente descartada pelo EDCB;
2. O federado síncrono executou o evento recebido pela mensagem correspondente à anti-mensagem. Neste caso, a co-simulação está comprometida (inconsistente). O trabalho de projeto do modelo precisa então ser revisto, e a simulação reinicializada.

O DCB também pode utilizar o recurso das ‘mensagens nulas’ para evitar situações de impasse [NOW99]. Embora nos experimentos realizados no escopo desta tese as mensagens nulas não tenham sido utilizadas, a implementação do protótipo do DCB possui uma primitiva que prevê seu uso.

A execução de um modelo de co-simulação pode gerar situações inconsistentes (situações de erro) que não podem ser resolvidas pelo DCB e embaixadores. Elas podem ocorrer devido a erros na construção do modelo de co-simulação ou por falhas de comunicação. Neste caso, estas situações precisam ser identificadas e comunicadas ao projetista. Para isso, é relevante que o DCB incorpore um conjunto de mensagens de erro de execução que auxiliem o projetista na detecção de problemas no modelo.

3.3.2.3 Paradas temporárias durante o exercício de uma simulação com vistas à depuração/visualização

Os bloqueios temporários da execução de uma simulação podem ser úteis para o acompanhamento ou visualização, depuração e manutenção de um modelo de co-simulação. A princípio, numa co-simulação temporizada, os bloqueios poderiam ser realizados através do controle sobre a evolução do GVT. Ou seja, ao frear a evolução do GVT os federados automaticamente ficariam impedidos de avançar. Contudo, existem implicações de acordo com o tipo de combinação existente entre federados em relação ao tipo de sincronização que utilizam.

Para FNT (Federado *NoTime*):

- FNTs passivos (que apenas recebem mensagens) têm sua evolução diretamente dependente do recebimento de mensagens. Deste modo, a simples interrupção do envio de mensagens faz com que o federado interrompa sua evolução até o restabelecimento da troca de mensagens;
- FNTs ativos (que apenas enviam mensagens) têm sua evolução independente da federação da qual participam. Este tipo de federado, quando disparado, evolui até que uma condição interna provoque seu bloqueio (independentemente do estado da federação e do valor do GVT). Federados *notime* utilizam o *timestamp* apenas para ordenar a entrega de mensagens, desconsiderando a sincronização no tempo;
- Se uma federação possui todos os federados do tipo FNT, a consistência da sincronização em uma federação continua mantida pelo DCB através do uso do tempo de evento das mensagens com propósitos de ordenação. No entanto, como federados *notime* não dependem do tempo global (apenas do *timestamp* para ordenação), a ocorrência de eventos em federados *notime* não provoca avanços no GVT. Por isso, o uso de bloqueios temporários baseados na interrupção do avanço do GVT (como descrito nos itens anteriores) não é possível. O gerenciamento de federações com todos os federados do tipo *notime* carece de mais estudos no gerenciamento da sincronização e no uso de bloqueios.

Para FS (Federado Síncrono):

- O bloqueio da evolução do GVT automaticamente bloqueia a evolução dos federados síncronos, já que a atualização de seu LVT é totalmente dependente do tempo global. Isso ocorre para ambos, federados síncronos ativos e passivos.

Para FA (Federado Assíncrono):

- Como FAs não têm o GVT como limitador de tempo para evolução, o simples bloqueio do tempo global não é suficiente para provocar uma parada na evolução deste tipo de federado, seja ele passivo ou ativo. Neste caso, o uso de registros dos eventos da simulação, ou mecanismos

que permitam controlar explicitamente a ocorrência de cada evento, podem ser utilizados para avaliação/acompanhamento do comportamento. No protótipo atual, não existe implementação para simulação otimista.

Em relação ao monitoramento do estado de uma simulação em tempo de execução (também sem o uso de recursos de bloqueio), é desejável a existência de uma interface mínima que permita a visualização de eventos em tempo de execução. O protótipo atual não incorpora funcionalidades com propósitos de monitoramento. No entanto, ao adicionar tal funcionalidade, a possibilidade de redução de flexibilidade do Tangram/DCB aumenta na medida em que a criação de mecanismos realmente genéricos de monitoramento é limitada pela diversidade de federações. Ou seja, na prática, algo além de um visualizador de mensagens pode exigir grande esforço de desenvolvimento de uma solução genérica na perspectiva do DCB.

Por isso, a melhor alternativa é atribuir para federados as tarefas de monitoramento de uma federação, quando necessário. Ou seja, incluir na federação um ou mais federados apenas com propósitos de monitoramento ou até mesmo de interatividade. Deste modo, um federado com propósitos de monitoramento é visto como um federado qualquer, cuja funcionalidade interna não interessa. Basta que o DCB seja configurado para suprir adequadamente os atributos de entrada e saída desse federado.

3.3.3 Suporte à comunicação

Além das funções de gerenciamento, o DCB implementa primitivas de suporte à comunicação entre nodos remotos e entre federados que estão sendo executados em um mesmo nodo. O NDCB é o módulo que incorpora as primitivas de comunicação.

A arquitetura do DCB estabelece uma separação entre as funções de gerenciamento de simulações (nos embaixadores) e as primitivas de envio e recebimento de mensagens (no NDCB). Esta característica facilita a troca de tecnologia de suporte à comunicação na medida em que as alterações necessárias para isso são restritas ao NDCB.

Deste modo, por exemplo, uma implementação do DCB pode utilizar *sockets* para comunicação remota e na mesma simulação uma chamada direta a um método para comunicação entre federados de um mesmo nodo. Estas alternativas foram utilizadas na implementação do protótipo do DCB.

A partir das considerações gerais desta seção, as seções a seguir abordam as características internas de cada um dos módulos da arquitetura do DCB: *gateway*, EF, EDCB, e núcleo do DCB.

3.4 *Gateway*

Um dos desafios mais expressivos na co-simulação é construção de um mecanismo consistente para a comunicação entre interfaces heterogêneas [BOR2000]. À luz deste desafio, a arquitetura do DCB elimina a necessidade de que federados

origem reconheçam a interface de federados destino. Para isso, encapsula os serviços de tratamento de interface no módulo denominado de *gateway* [STR98]. Este módulo, cuja arquitetura é representada na Figura 3.4, provê a comunicação local entre federado e respectivos embaixadores.

Especificado para preservar a integridade tanto do federado quanto do DCB, o *gateway* é o único módulo que precisa ser modificado ao ser adicionado um novo federado a uma federação. Embora ainda que sejam necessárias modificações no *gateway*, a concentração de esforços num único módulo de interface tende a reduzir expressivamente o custo da adição de novos federados.

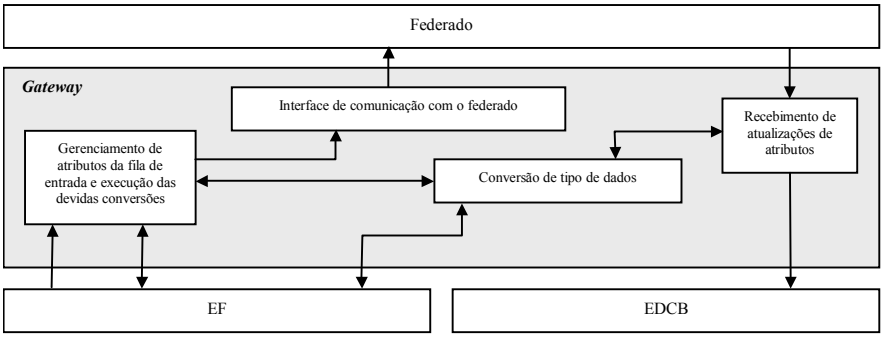

Figura 3.4: Arquitetura do Gateway

A inexistência de um módulo intermediário para tradução/tratamento de interface resultaria na necessidade de modificações internas no federado, o que não é desejável, e até inviável no caso de uso de IP's, por exemplo. Por outro lado, a realização de modificações no ambiente de suporte à co-simulação poderia resultar em custos maiores ainda se comparado a alterações no federado. Em ambos os casos, é visível a necessidade de maior esforço de programação/reestruturação se comparado ao uso do *gateway*.

Semelhantemente ao que é discutido em [NIC2001], onde o projetista trata separadamente os requisitos de interface e de comunicação, o *gateway* permite que o projetista se preocupe apenas com a interface do federado, sem se preocupar com aspectos de comunicação e de interface de simulação.

Para a adição de novos federados, o DCB prevê a geração automática de *gateway*s através da configuração de federações (discutida no próximo capítulo). Para isso, o ambiente em que o DCB está inserido tem a capacidade de oferecer *templates* (moldes) pré-definidos. Eles podem ser utilizados na geração automática de *gateway*s para federados cujas características estejam previstas nesses *templates*.

Contudo, é muito difícil manter a generalidade quando não existe previsão (ou definição) das características de um novo federado, mesmo com o uso de *templates*. Como a proposta do DCB objetiva permitir que qualquer federado seja adicionado a uma federação, é difícil medir o quanto a implementação do *gateway* pode ser automatizada. Embora a arquitetura do DCB (Figura 3.1) faça com que apenas o *gateway* tenha contato com os atributos de interface de um federado, ainda podem ser necessárias alterações no código do *gateway*. Em ambas as situações, a parte do código que trata das operações específicas (comportamento) dos federados é preservada na íntegra.

Por exemplo, se um federado está implementado numa linguagem reconhecida pelo DCB, chamadas diretas aos métodos do federado podem ser feitas. Ou se um federado utiliza primitivas de comunicação tais como *sockets*, pode ser construído um *template* que reconheça tais primitivas. Outras alternativas tais como o uso de arquivos, ou a codificação de *wrappers* [YOO2001], também podem ser previstas pelos *templates*.

Caso seja necessário adicionar um federado cuja interface não seja prevista pelos *templates*, a modificação manual do *gateway* pode ser feita pelo projetista da federação. Embora, neste caso, o *gateway* não seja gerado de forma automática por completo, a preservação da integridade do federado e do DCB se mantém.

No provimento das tarefas de gerenciamento de interface entre federado e embaixadores, o *gateway* possui quatro principais atribuições [SPE2003]:

- Receber do federado as atualizações de valores de atributos de saída da interface;
- Conversão de tipo de dados de acordo com configurações disponíveis no embaixador do federado (tradução);
- Nos casos em que dois federados que se comunicam estiverem descritos em níveis diferentes de abstração, o *gateway* realiza conversão semântica de dados recebidos antes de repassá-los ao federado;
- Mediante notificações dos embaixadores, repassa ao federado atualizações em seus atributos de entrada devido ao recebimento de mensagens de outros federados.

3.5 Embaixadores

O termo ‘embaixador’ é utilizado para designar os módulos que são responsáveis pela execução das atividades de gerenciamento das mensagens recebidas e enviadas por cada um dos nodos que comportam um federado. Ao contrário de HLA, onde o termo ‘embaixador’ é designado para indicar a parte de código interna ao federado que é responsável pelas chamadas à RTI, no DCB os embaixadores são invisíveis aos federados. O código dos embaixadores é estático, mudando apenas a sua configuração para diferentes interfaces. O *gateway* é o único módulo da arquitetura do DCB que sofre alterações no código, geralmente pouco expressivas.

Os embaixadores são responsáveis pelo estabelecimento das relações funcionais entre federados de uma federação e pelo gerenciamento das mensagens trocadas entre estes federados. Basicamente, as relações funcionais têm o propósito de fazer uma ligação entre atributos de saída de um federado com atributos de entrada de um segundo federado destino, seja ele da mesma federação ou de uma outra federação. Os federados desconhecem completamente a existência dos demais federados da federação de que participa. Esta independência permite que federados sejam inseridos e retirados de uma federação em tempo de execução.

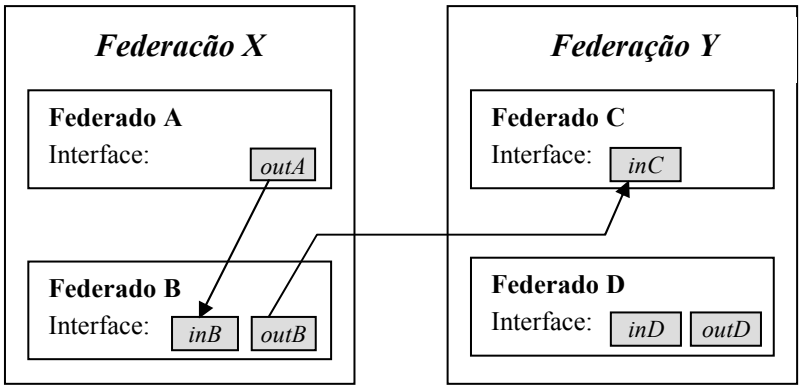

Figura 3.5: Ligação entre atributos de federados (configuração de federação)

Tomando como exemplo a Figura 3.5, um atributo de saída *outA* de um federado A pertencente a uma federação X possui uma ligação com um atributo de entrada *inB* de um federado B da mesma federação. E o federado B possui um atributo *outB* ligado com um atributo *inC* de um federado C que pertence a uma federação Y. Neste caso, os embaixadores do federado A não mantêm nenhum registro sobre a relação existente entre B e C.

As seções a seguir apresentam as funcionalidades de cada um dos embaixadores, EF e EDCB.

3.5.1 Embaixador do Federado (EF)

O Embaixador do Federado (EF) tem como objetivo geral receber as mensagens do NDCB (que executa operações de comunicação) e enviá-las ao federado, via *gateway*, de forma consistente no tempo de simulação.

Para isso, o EF executa um conjunto de atividades relacionadas ao tratamento de mensagens e ao tratamento de tempo. Ambas as atividades possuem uma relação de dependência na medida em que a ordem de envio de mensagens ao federado, sob controle do EF, depende do tempo local do federado e do tempo em que um evento informado por cada mensagem deve ser executado. A Figura 3.6 apresenta a arquitetura do EF.

O EF não sofre modificações de código quando alocado para diferentes federados, mas as informações de configuração são alteradas de acordo com o federado com que interage. Para isso, cada embaixador implementa uma estrutura de dados dinâmica para adaptação a diferentes configurações. São exemplos de itens considerados no registro de configuração: identificação do federado e da federação, atributos de entrada, tipo de sincronização do federado, número do *Internet Protocol* do nodo em que está o federado, porta de comunicação, entre outros.

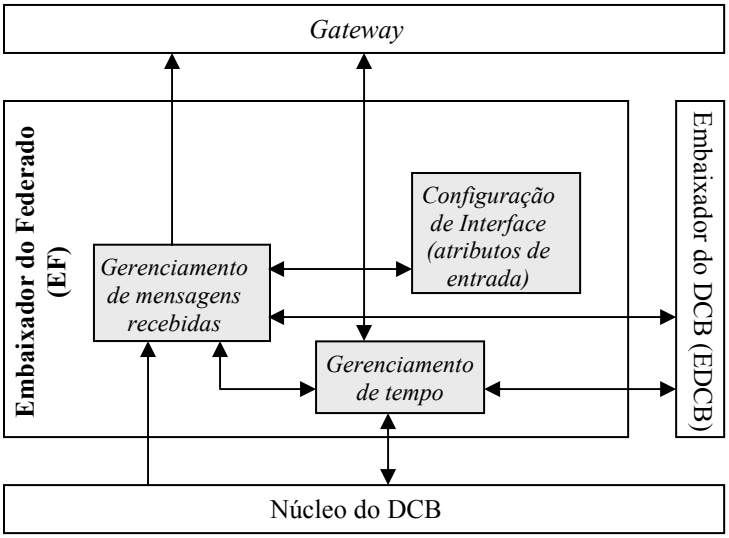

Figura 3.6: Funcionalidades do Embaixador do Federado

São atribuições básicas do EF: manter registro sobre a configuração de atributos do federado, gerenciar uma fila de mensagens recebidas de outros federados e repassá-las ao federado de modo consistente em relação ao LVT e ao tempo de evento das mensagens.

3.5.1.1 Gerenciamento de mensagens

No contexto do gerenciamento de mensagens, o EF é responsável por realizar a decodificação das mensagens recebidas de outros federados. Contudo, o EF não realiza operações remotas. As mensagens são recebidas por intermédio das primitivas de comunicação do NDCB.

No trabalho de decodificação das mensagens recebidas, o federado identifica o atributo destino, seu tipo, entre outras informações de controle, e mantém a mensagem em uma lista de espera. A mensagem somente será repassada ao federado quando houver consistência em relação ao tempo. São três situações distintas para controle de consistência:

- Federado síncrono: a mensagem só é enviada ao federado quando o GVT igualar ou ultrapassar o valor do tempo de evento da mensagem;
- Federado *notime*: a mensagem pode ser enviada ao federado assim que postada na lista de entrada mantida pelo respectivo EF. No entanto, a ordem em que as mensagens devem ser tratadas precisa ser fiel à ordem de emissão a partir dos federados origem. Para isso, o EF utiliza o valor do *timestamp* apenas com propósitos de ordenação;
- Federado assíncrono: a mensagem é enviada ao federado assim que recebida pelo EF. Esta política exige a presença de algoritmos de retorno para tratamento de violações de LCC. A Seção 2.2 deste texto identifica estudos (e referências) sobre algoritmos de recuperação de estados seguros em simulação distribuída otimista.

Buscando flexibilizar e facilitar o trabalho de tratamento de mensagens, não apenas do EF, mas também do EDCB e do NDCB, todos os dados enviados através dos atributos de saída dos federados são traduzidos pelo *gateway* para o tipo *string*. No federado destino, as *strings* são traduzidas pelo *gateway* respectivo para o tipo de dado esperado por este federado. Esta informação estará disponível nas configurações mantidas pelo DCB do federado destino.

A opção pelo uso de apenas um tipo de dado para empacotamento e envio de valores de atributos (mensagens) é justificada pelo fato de que o EF não executa nenhuma tarefa remota. Por isso, incumbi-lo de traduzir os valores de atributos para o formato esperado pelo federado destino pode exigir expressivo esforço adicional na sua implementação, tornando-o mais complexo e provavelmente mais instável.

3.5.1.2 Gerenciamento de tempo no EF

O EF também é responsável pelo gerenciamento do tempo local de simulação do federado. Para isso, o EF possui dois principais mecanismos: é sensível às atualizações do GVT; e limita a atualização do LVT, sempre que solicitada por federados síncronos, ao valor do GVT síncrono.

Inicialmente, o EF deve ser sensível às modificações do GVT. A cada vez que o GVT é atualizado, o EF verifica se existem mensagens originárias de outros federados cujo tempo de evento é menor que o novo GVT. Estas mensagens são então enviadas ao federado por intermédio do *gateway*.

Esta política é adotada para federados síncronos e para federados *notime*, o primeiro sob o controle do GVT síncrono. No caso de federados assíncronos, a restrição é de que o tempo de evento não seja menor do que o GVT assíncrono. Deste modo, toda mensagem trocada entre federados assíncronos é imediatamente repassada ao federado destino pelo EF (sempre via *gateway*).

No segundo mecanismo, de atualização do LVT, o EF também executa operações diferenciadas de acordo com o tipo de sincronização para o qual o federado foi configurado. Para federados síncronos, o EF é sensível à atualização do tempo local feita pelo federado. Ou seja, é o federado que decide quando e para quanto evolui o valor do LVT. Contudo, cabe ao EF evitar que o novo valor do LVT ultrapasse o valor do GVT síncrono.

O atendimento deste requisito é visto como um desafio para a execução síncrona de federados. A dificuldade reside no fato de que um federado síncrono pode incrementar seu LVT (evoluir no tempo) sem troca de mensagens com a federação. Deste modo, não haveriam empecilhos para que seu LVT ultrapasse o valor do GVT, podendo gerar situações de erro. Este desafio é particularmente expressivo ao adicionar um componente de hardware, por exemplo, na federação.

Para evitar que o LVT de federados síncronos ultrapasse o valor de GVT, as operações de incremento do LVT são controladas pelo EF em parceria com o NDCB. Para isso, o EF compara todos os pedidos de avanço do LVT com o valor do GVT. Se o avanço pretendido ultrapassa o GVT, o EF mantém o LVT igual ao GVT. Caso contrário, o avanço é autorizado na íntegra pelo EF. Isto impede automaticamente o avanço do LVT de federados síncronos para estados inseguros. Um estado é dito

inseguro quando existe a possibilidade de que um federado receba mensagens com tempo de evento menor que seu LVT [FER95].

Para federados *notime*, o EF assume toda a responsabilidade sobre o controle do LVT mantendo seu valor sempre igual ao GVT síncrono. Para cada mensagem enviada por um federado *notime*, o EDCB incrementa o atributo que armazena o tempo de evento (*timestamp*) a partir do valor do GVT. Assim, automaticamente é mantida a ordem entre mensagens com tempos de evento distintos.

Federados assíncronos recebem do EF um tratamento semelhante ao oferecido para federados síncronos. No entanto, as solicitações de avanço do LVT feitas pelo federado não são restringidas pelo EF.

3.5.2 Embaixador do DCB (EDCB)

O EDCB tem como objetivo geral prover os mecanismos necessários para receber solicitações de envio de mensagens transmitidas pelo *gateway*, empacotar/codificar as informações, e enviá-las ao destino através dos serviços de comunicação do NDCB. O *gateway* dispara tais solicitações a partir das alterações do valor dos atributos de saída da interface do federado que ele gerencia. Este serviço é representado na Figura 3.7 pelo módulo de "Gerenciamento de mensagens enviadas".

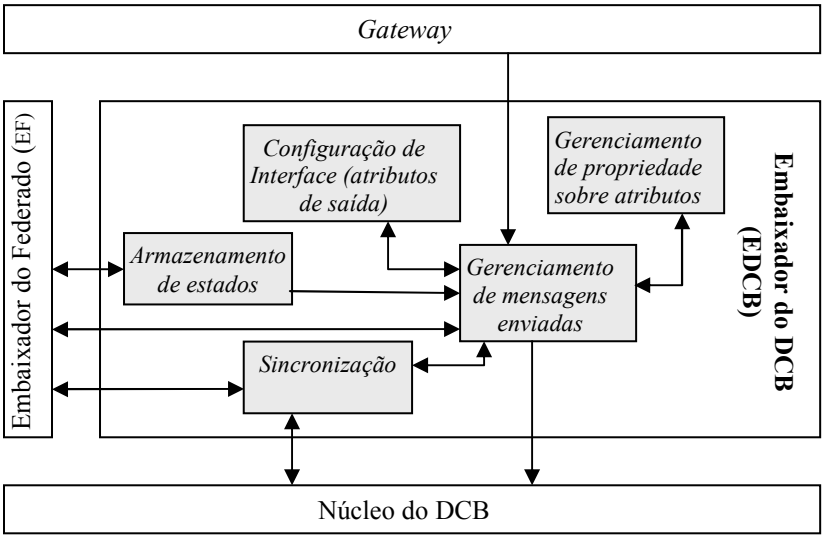

Figura 3.7: Funcionalidades do Embaixador do DCB

Na tarefa de configuração, assim como no EF, o EDCB mantém o registro das informações que determinam seu comportamento perante o federado e a federação. Algumas informações, tais como identificação de federado e federação e tipo de sincronização, são compartilhadas entre ambos os embaixadores. Contudo, é de interesse particular do EDCB manter registro das informações sobre atributos de saída do federado, pois é no EDCB que são realizadas as operações de empacotamento e envio (via NDCB) de mensagens.

O gerenciamento de mensagens possui uma dependência direta do módulo de gerenciamento de propriedade sobre atributos. Este módulo implementa uma 'exclusão mútua' no uso de atributos de entrada que são compartilhados por federados distintos. Por exemplo, se dois ou mais atributos de saída possuem ligação com um mesmo atributo de entrada no federado destino, atribuições simultâneas de valor a este atributo podem degradar o dado de um dos atributos origem. Para resolver este problema, o módulo de gerenciamento de propriedade não permite o envio de mensagens caso o federado origem não possua a propriedade do atributo destino.

Este controle é importante para manter o uso exclusivo (exclusão mútua) de um atributo de entrada quando múltiplos atributos de saída compartilham um mesmo atributo de entrada.

O armazenamento do histórico de mensagens enviadas também faz parte dos objetivos do EDCB. Este histórico é utilizado para a execução de operações de retorno a estados seguros perante a ocorrência de erros provocados por violações de LCC. A política de retorno no tempo para um estado seguro é prevista apenas para federados configurados para o modo assíncrono.

Embora o gerenciamento de tempo seja realizado principalmente pelo EF, o EDCB desenvolve algumas operações que utilizam o tempo em duas situações distintas (propósitos de sincronização): no empacotamento de mensagens e no armazenamento de estados seguros.

Na primeira situação, por exemplo, o EDCB utiliza o valor do GVT como referência para ordenar as mensagens enviadas por federados *notime*. Na segunda, o EDCB pode utilizar tanto o valor do LVT, bem como do GVT, para armazenar estados seguros com propósitos de retorno.

3.6 Núcleo do DCB (NDCB)

O Núcleo do DCB (NDCB) tem como propósito geral atender às solicitações de comunicação entre federados. Embora o NDCB utilize informações de configuração para realizar o roteamento de mensagens, não faz parte dos seus objetivos interferir em atividades de gerenciamento em termos de evolução da simulação (dados e tempo). Ou seja, as tarefas do NDCB se limitam ao oferecimento de primitivas para o simples envio e recebimento de mensagens, desconhecendo seu conteúdo.

As mensagens podem ser de caráter administrativo ou de simulação. Mensagens administrativas são utilizadas especificamente pelo DCB para execução de atividades de gerenciamento. As mensagens de simulação transportam valores, com origem e destino definidos na configuração de federações, que são emitidos/recebidos nos atributos de interface dos federados (via *gateway*).

Para atender solicitações de envio de mensagens feitas pelo EDCB, o NDCB pode oferecer serviços diferenciados para comunicação com federados remotos e com federados locais (Figura 3.8). Para federados remotos é inerente o uso de serviços de comunicação remota (*sockets*, RPC, CORBA). Por outro lado, na comunicação com federados locais podem ser utilizadas alternativas de comunicação direta de acordo com a técnica/linguagem de implementação utilizada (compartilhamento de memória, chamada explícita a métodos).

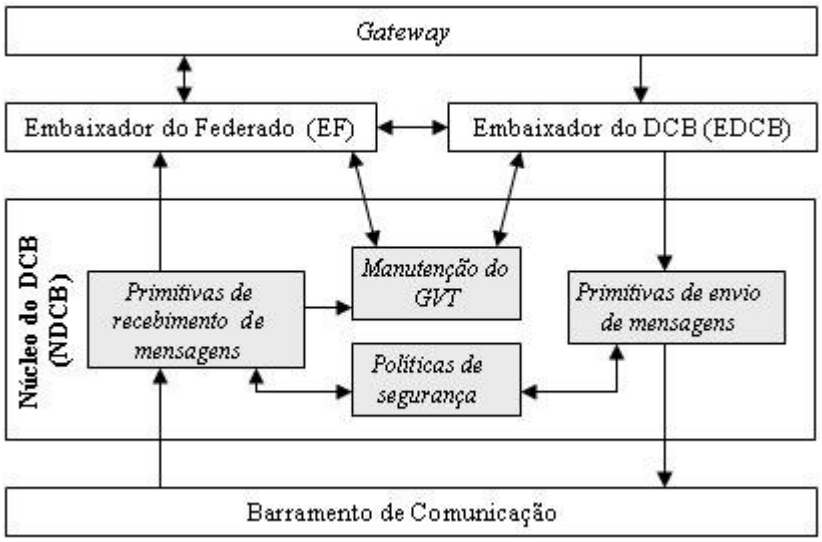

Figura 3.8: Funcionalidades do Núcleo do DCB

Para atender solicitações de envio de federados remotos (origem), o NDCB oferece serviços cuja tarefa é encaminhar ao EF local as mensagens recebidas. Particularmente, a tarefa de recebimento de mensagens exige pouco esforço do NDCB.

O NDCB também prevê um módulo que implementa políticas de segurança voltadas para o exercício de simulações e para proteção de componentes com restrições vinculadas a questões de IP. Por exemplo, queda de um nó, restrições de acesso a informações enviadas e recebidas de federados, entre outros. O estudo de políticas de segurança não faz parte do escopo desta tese, sendo citado como perspectiva futura nas considerações finais.

3.7 Contribuições do DCB

A evolução dos sistemas heterogêneos em termos de variedade, complexidade e tamanho faz com que este tipo de sistemas dificilmente siga um mesmo padrão, tecnologia de implementação, ou nível de abstração. Esta característica é refletida na construção de modelos de representação para sistemas heterogêneos, tornando-se mais difícil de manter a fidelidade do modelo em relação ao sistema real [CHE2001].

Esta realidade fornece justificativas suficientes para investimentos no desenvolvimento de pesquisas voltadas para a concepção de soluções mais flexíveis e genéricas direcionadas à simulação de sistemas heterogêneos. No escopo desta tese, o esforço foi direcionado para simulação heterogênea em ambiente distribuído, promovendo a distribuição física dos componentes cooperantes que pertençam a um mesmo modelo.

As contribuições desta tese em busca de flexibilidade e generalidade no suporte a simulações heterogêneas distribuídas foram concentradas na proposta do DCB, apresentado neste capítulo. Não faz parte dos objetivos do DCB promover contribuições sobre as técnicas de simulação, por exemplo na otimização de algoritmos de

sincronização, de retorno a estados seguros e de comunicação, entre outros. A bibliografia apresenta um conjunto de referências expressivo a respeito destas técnicas. Contudo, a busca de estratégias eficazes no uso de tais técnicas para o suporte à construção e execução distribuída de modelos heterogêneos é vista como desafio à luz do estado da arte deste tema [PAG2000] [HUB98].

O DCB, então, é proposto nesta tese como uma estratégia alternativa de suporte à co-simulação distribuída observando quatro itens principais: a distribuição física; a independência dos federados; o encapsulamento das estratégias de gerenciamento de tempo, de dados e de comunicação; e a sincronização híbrida. O atendimento de modo integrado (conjunto) destes itens promove os conceitos de interoperabilidade e de substituição (*interchangeability*) [CHE2001b] entre tipos diferentes de componentes (hardware, software, partes reais), entre componentes descritos em diferentes linguagens de modelagem, entre componentes desenvolvidos separadamente com propósitos diferenciados, entre outros.

O oferecimento de mecanismos flexíveis, não proprietários e de modo integrado, que permitem adicionar facilidade e agilidade no tratamento de questões de interoperabilidade e *interchangeability* em simulação heterogênea define o escopo das contribuições deste trabalho de tese.

3.7.1 Distribuição

Embora a simulação distribuída seja alvo de variados estudos, mais recentemente a composição de componentes heterogêneos (co-simulação) com propósitos de execução cooperativa distribuída tem despertado maior interesse. Na simulação heterogênea, além de aspectos de envio e recebimento de mensagens, a interoperabilidade entre componentes também precisa ser gerenciada em conjunto com a execução de primitivas de comunicação. Esta característica da simulação heterogênea adiciona complexidade às soluções de suporte à execução de modelos distribuídos. Além disso, estudos em co-simulação são principalmente voltados para execução centralizada.

Neste contexto, para a execução distribuída de federações heterogêneas, o DCB explora avanços em simulação distribuída no escopo da co-simulação (hw/sw). Este aspecto é relevante como contribuição na medida em que ainda há espaço para uma maior intersecção entre o estado da arte em simulação distribuída e as necessidades de validação em fases intermediárias do projeto de sistemas heterogêneos, por exemplo sistemas embarcados.

3.7.2 Independência de federados

Quanto à independência de federados, a arquitetura do DCB tem como prioridade a preservação da sua integridade. Para isso, o primeiro princípio é de que o federado visualiza um único módulo externo com quem se comunica (o *gateway*), tratando apenas aspectos de interface. Ou seja, o federado permanece alheio às operações de gerenciamento da federação que ocorrem além do *gateway*. Elas ocorrem no EF, no EDCB e no NDCB, que por sua vez desconhecem o federado, pois visualizam apenas as primitivas do *gateway* para atualização de valores de atributos. Esta característica pode ser observada na Figura 3.1 no início deste capítulo.

Pelo mesmo motivo, na visão do *gateway*, também é verdadeira a afirmação de que o *gateway* visualiza apenas as primitivas do federado utilizadas na alteração de valores de atributos de entrada. E, para atualização de atributos de saída, o federado é quem conhece a primitiva correspondente oferecida no *gateway*. Tais medidas promovem a independência da estrutura interna dos federados e da estrutura (ferramenta) de suporte à simulação distribuída.

Num comparativo com HLA, tomando como referência uma federação de um restaurante disponível em [KUH2000], os federados possuem em torno de 30% do seu código implementado de forma específica para execução de chamadas a primitivas RTI. Além disso, estas chamadas exigem o conhecimento das operações de gerenciamento de tempo e de dados da RTI, além do conhecimento de informações de outros federados com quem o federado origem precisa se comunicar. Esta característica exige alterações na estrutura interna do federado, inviabilizando sua preservação e consumindo esforço de programação. Ou seja, sem acesso completo ao código do federado não há como integrá-lo a uma federação.

Nesta mesma situação exemplo, no entanto com o uso da proposta do DCB, se um federado possui, por exemplo, 5 atributos de saída, bastariam 5 chamadas a uma mesma primitiva do *gateway*, não importando o tamanho do federado. Deste modo, as alterações, que são mínimas, se restringem ao tratamento de interface do federado, preservando totalmente a integridade de seus detalhes internos (comportamento). O trabalho de configuração da federação, realizado sem a necessidade de alterações internas aos federados, é o fator principal que viabiliza a independência dos federados. Quem absorve tais informações de configuração são os módulos do DCB, essencialmente os embaixadores.

Um outro fator positivo, resultante desta característica do DCB, é a redução do tempo necessário para a construção de uma federação. A eliminação da necessidade de alterações em detalhes internos do código dos federados, substituída pela política de configuração e geração de *gateway*s, reduz potencialmente o esforço de modelagem. Mesmo se necessário interferir no código do *gateway* para suprir situações não previstas nos mecanismos de geração dos *gateway*s, a tendência de redução do esforço permanece expressiva. Justifica-se esta afirmação pelo fato de que o *gateway* mantém a mesma estrutura interna para qualquer federado, onde apenas as ações que representam a comunicação com os federados sofrem alteração. Isto, se a tecnologia do federado não for prevista no módulo de configuração que gera os *gateway*s (nos *templates*). Se prevista, interferências no código do *gateway* não são mais necessárias.

A idéia de separação entre detalhes internos de componentes e de aspectos de comunicação, estratégia inerente aos fundamentos de concepção do DCB, vem ganhando espaço nas pesquisas em torno da co-simulação. Observa-se também o direcionamento de esforços para estudos que abordam metodologias e técnicas que valorizam o aumento do nível de abstração no projeto de sistemas [KEU2000]. Esta tendência é indicador de que há mérito justificável na busca de alternativas em busca de interoperabilidade entre componentes descritos em níveis distintos, como explorado no escopo do Tangram/DCB.

3.7.3 Encapsulamento das políticas de gerenciamento

O encapsulamento de estratégias de gerenciamento mantido pelo DCB está diretamente ligado aos propósitos de independência dos federados. A completa separação entre detalhes internos dos federados e as políticas de gerenciamento de dados e de tempo, entre outros, necessárias à execução de um modelo heterogêneo e distribuído de simulação, promove tal independência. Esta característica contribui grandemente com o reuso de componentes.

3.7.4 Sincronização híbrida

Em relação à sincronização, o DCB permite que federados síncronos, assíncronos e *notime* possam participar simultaneamente de uma mesma federação. Em geral, pesquisas em simulação distribuída concentram esforços no provimento de cooperação entre federados com as mesmas características no tratamento de tempo, essencialmente em simulações homogêneas. Este esforço do DCB é baseado na tendência de que uma melhor relação custo/benefício pode ser alcançada com o uso de modelos híbridos em relação ao gerenciamento de tempo na execução de modelos de simulação [YOO2000].

Por exemplo, modelos síncronos não permitem violações de LCC, contudo os federados não têm autonomia para avançar no tempo, freando a sua evolução. Nos modelos assíncronos os federados podem avançar no tempo sem restrições reduzindo o tempo total de simulação, porém podem provocar violações de LCC. Ao combinar ambas as formas de sincronização é possível usufruir parcialmente das vantagens de ambos, melhorando o resultado final.

A partir da especificação do DCB, o próximo capítulo discute requisitos para a construção de ferramentas cujo objetivo geral é o oferecimento de suporte ao trabalho de modelagem e construção de federações heterogêneas, fases anteriores à execução distribuída. Tais requisitos têm como propósito promover a continuidade do trabalho em direção à concepção de um ambiente que abrange desde a identificação e busca de componentes até a avaliação dos resultados de execução.

4 IDENTIFICAÇÃO DE REQUISITOS DE UM AMBIENTE PARA A MODELAGEM DE FEDERAÇÕES

4.1 Introdução

A variabilidade de tecnologias que podem ser utilizadas em modelos heterogêneos distribuídos dificulta seu enquadramento em um determinado padrão, ou conjunto de características. Como discutido no capítulo anterior, este fator adiciona complexidade às tarefas de gerenciamento/suporte à cooperação entre componentes do modelo. No entanto, as dificuldades não são restritas à execução, mas interferem também no trabalho de construção de modelos (modelagem) em relação a disponibilidade, acesso e integração entre componentes heterogêneos. A dificuldade de reuso de componentes também aumenta de acordo com a redução do nível de abstração (proximidade com a implementação) utilizado por um projetista [PAG2000].

O desenvolvimento de mecanismos ou ambientes de projeto que combinam facilidades tais como busca, catalogação e integração de componentes com propósitos de modelagem heterogênea pode ajudar a reduzir tais dificuldades. Atualmente a concepção de novas metodologias voltadas para a construção de modelos de co-simulação também tem despertado interesse, por exemplo usando o projeto colaborativo distribuído.

Segundo esta tendência, este capítulo apresenta um levantamento de requisitos de um ambiente de modelagem e construção de federações à luz da arquitetura do DCB. Os requisitos discutidos neste capítulo tem como objetivo geral identificar mecanismos de apoio para a construção de modelos heterogêneos e para sua execução distribuída suportada pelo DCB.

Por outro lado, não faz parte dos objetivos deste levantamento de requisitos considerar a construção de novos federados, nem a identificação de mecanismos de comunicação entre diferentes ferramentas de desenvolvimento. A Figura 4.1 apresenta os módulos previstos para o ambiente. Os módulos sombreados foram implementados no escopo do DCB, apresentado no capítulo anterior.

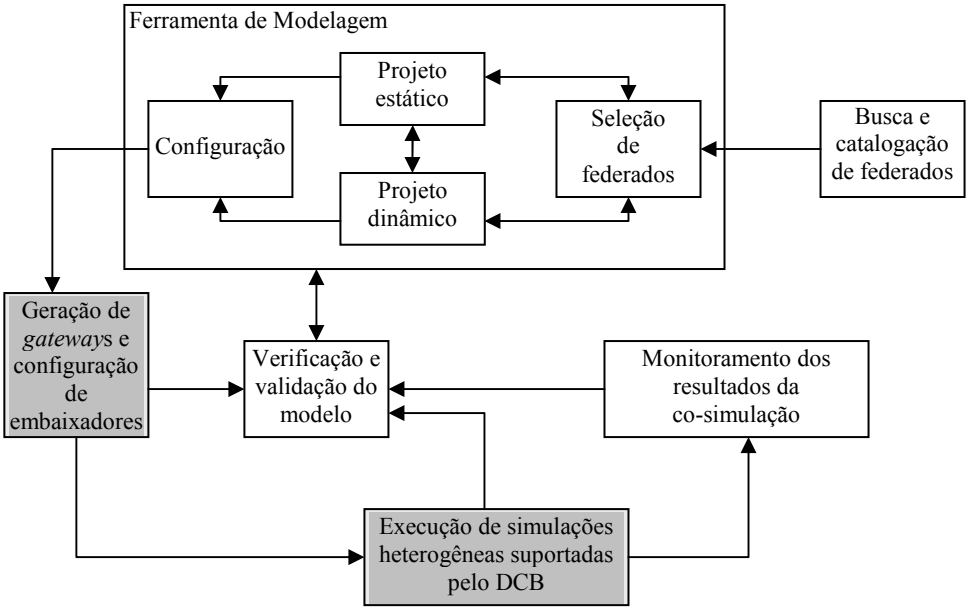

Figura 4.1: Módulos previstos para o ambiente

As seções a seguir apresentam uma discussão limitada aos requisitos para cada um dos módulos previstos. Não faz parte dos objetivos deste capítulo identificar ou propor mecanismos, técnicas ou soluções para os módulos.

A denominação 'Tangram' é utilizada para o ambiente que integra o DCB e os demais módulos previstos.

4.2 Busca e catalogação de componentes

O uso de ferramentas de software com o objetivo de facilitar o trabalho de projeto/construção de modelos de co-simulação tende a reduzir o tempo de projeto e de desenvolvimento, bem como contribuir com a correção. Contudo, mesmo tendo à disposição as facilidades de um ambiente (ou ferramenta) desse tipo, o fato de que os modelos são construídos por composição [CHO98a] de federados existentes reflete na necessidade de localizar e identificar previamente federados com propriedades específicas, de acordo com as partes de um modelo em construção, para que possam ser reutilizados numa federação [FAU99].

O tempo gasto na tarefa de identificação de federados que possam ser úteis para uma federação em construção pode ser reduzido com a ajuda de ferramentas de apoio. Este fato justifica a construção de um módulo no Tangram cujo objetivo é a busca e catalogação de federados existentes. Tendo à disposição (catalogadas) informações sobre múltiplos federados, torna-se menos complexa e mais rápida a tarefa de identificação de federados cujas propriedades sejam adequadas para os propósitos de um modelo de co-simulação em construção. O princípio básico deste módulo é manter o registro das propriedades de um federado, esteja ele disponível localmente ou não. O

trabalho de registro de propriedades precisa levar em consideração itens tais como segurança, acesso, requisitos de uso, interface, entre outros.

A princípio, a busca de federados pode ser baseada na comparação entre as propriedades desejáveis dos federados definidos/especificados pelo projetista durante o projeto de um modelo de co-simulação e as propriedades de federados existentes, sejam eles remotos ou locais. Para isso, um requisito importante para este módulo é a capacidade de manter o registro das propriedades de federados existentes. O conjunto de registros pode ser denominado de repositório de federados. Um repositório pode conter o próprio federado (físico), ou apenas informações dos federados.

Na atividade de busca e catalogação, a heterogeneidade inerente aos federados existentes implica na observação de características que beneficiam o reuso. São exemplos dessas características [REU99]:

- Se o federado foi construído com propósitos específicos, ou a possibilidade de reutilizá-lo foi considerada durante seu desenvolvimento. Quando existe interesse no reuso, existe uma tendência de que na construção de um federado sejam adicionadas características não relevantes para o problema original que motivou a construção desse federado (geralmente no nível de interface);
- Conjunto de dados que o federado precisa receber do lado externo para seu correto funcionamento;
- Se, a partir de um federado inicial, existem outros federados compatíveis em tecnologia de desenvolvimento, ou que implementem extensões de seu comportamento interno. Mesmo num sistema heterogêneo, o aumento do número de federados 'compatíveis' reduz proporcionalmente o trabalho do projetista e a complexidade do sistema. Quanto maior a compatibilidade dos federados utilizados num mesmo modelo heterogêneo (federação), mais fácil e confiável será sua construção e execução;
- Se o federado implementa estrutura de dados e comportamento necessários aos propósitos de um sistema em desenvolvimento. Por exemplo, se uma determinada parte do sistema precisa evoluir no modo assíncrono de simulação, que tipo de operações de salvamento de estados e retorno são suportadas pelo federado [DAL2000];
- Aspectos de *Intellectual Property* (IP) devem ser levados em consideração no registro de propriedades de federados existentes.

Também é desejável considerar como requisito para a busca e catalogação de componentes o oferecimento de condições para a realização de projetos colaborativos. O fato de que diferentes projetistas podem trabalhar sobre um mesmo projeto, cujos federados também estejam dispersos geograficamente, sugere que o módulo de busca e catalogação ofereça, para todos os projetistas, repositórios de federados equivalentes.

4.3 Modelagem de federações por composição de federados

Baseado em abordagens composicionais [CHO98a] [CHO98b], o módulo de suporte à modelagem de federações tem como objetivo geral oferecer uma área de trabalho voltada para o projetista de modelos de co-simulação. É requisito principal deste módulo oferecer mecanismos suficientes para que o projetista possa exercer pleno controle sobre as etapas de projeto e de implementação de um modelo com características de heterogeneidade e distribuição. Para isso, é essencial a existência de repositórios descentralizados de federados, funcionalidade associada ao módulo de busca e catalogação de federados. Este módulo está em consonância com conceitos de modelagem e simulação visual interativa [WAG96].

A implementação deste módulo pode ser realizada em duas partes principais, a parte que trata do projeto estático e a parte que trata do projeto dinâmico de modelos de co-simulação, como sugerido na Figura 4.1. De modo geral, a parte estática de um projeto é desenvolvida com base na descrição das propriedades de cada componente, seja ele utilizado em uma federação ou não. São exemplos de propriedades: atributos, tipo de sincronização, tipo de acesso, entre outros.

Por outro lado, a modelagem da parte dinâmica tem como objetivo geral estabelecer a relação funcional entre os federados. Esta relação se dá através da troca de dados cujos critérios de comunicação são estabelecidos pela ligação entre atributos de origem e destino. Basicamente, estas ligações são a base da especificação de estratégias de cooperação entre os diferentes federados interconectados. A correção e agilidade na configuração das relações funcionais deve ser valorizada em ferramentas de modelagem.

Esta tarefa é feita através da definição do modo como cada federado pode alterar os atributos de sua interface. Ou seja, a partir de um federado origem, deve-se especificar para quais atributos de quais federados destino (um único ou múltiplos) um federado origem poderá enviar dados, e em que situações. Para isso, é necessária uma ligação permanente entre um atributo de saída de um federado origem e um ou mais atributos de entrada de um ou mais federados destino. Feita esta ligação, as estruturas internas do DCB (sincronização e gerenciamento de dados) se encarregam de manter a correção de uma simulação heterogênea em tempo de execução.

Ambos, projetos estático e dinâmico, interferem na atividade de seleção dos federados. Um federado existente pode ser selecionado para ser adicionado a um modelo de co-simulação desde que suas propriedades sejam adequadas aos propósitos deste modelo em construção. Para isso, uma estrutura de gerenciamento dessas propriedades precisa ser especificada. Nela, são considerados aspectos tais como:

- Identificação: cada federado que é adicionado a um modelo de co-simulação recebe uma identificação única;
- Atributos: descrição e identificação dos atributos de entrada e de saída do federado;
- Recursos necessários: particularidades a respeito da exigência de recursos (de software ou hardware) necessários ao correto funcionamento do federado;

- Funcionalidade: descrição do comportamento interno do federado (tipo de serviço que oferece);
- Sincronização: modo de execução em função do tempo. Se assíncrono, registrar se o federado possui suporte interno para retorno a estados seguros no caso de violação de LCC. Com isso o ambiente pode construir modelos de co-simulação híbridos, onde alguns federados executam no modo assíncrono e outros no modo síncrono, ou ainda no modo *'notime'*;
- Localização e execução: informações a respeito da localização física do federado e sobre o nível de permissão que o projetista possui para acesso ao nodo em que o federado estiver sendo executado.

O módulo de projeto pode também incorporar mecanismos de apoio para a ações de inclusão, exclusão, ou substituição de federados em tempo de execução de um modelo de co-simulação. Estas ações são previstas na arquitetura do DCB apresentada no capítulo anterior.

Numa substituição, o novo federado precisa estar com os atributos de interface (arquivo XML e *gateway*) configurados de forma equivalente à configuração do federado antigo. Ou seja, embora o federado possa ser substituído, a interface com os federados que permanecem em execução precisa ser preservada, caso contrário a execução destes federados também teria de ser interrompida para que a sua interface possa sofrer as adaptações necessárias (configuração) à cooperação com o novo federado. Este problema não acontece quando um federado é interrompido (na federação em execução) e disparado novamente.

Como uma segunda etapa de desenvolvimento do módulo de projeto, ele pode ser estendido para suportar o projeto colaborativo de modelos de co-simulação. Deste modo, múltiplos projetistas podem estar em locais físicos remotos trabalhando num mesmo modelo de co-simulação. A grande diferença, neste caso, é de que um federado visto como local por um projetista será visto como remoto pelos demais. Contudo, o mesmo modelo de co-simulação é visto por ambos os projetistas. Atualmente, o projeto colaborativo de sistemas embarcados tem sido alvo de pesquisas, como por exemplo em [IND2003] e de uma tese em andamento no contexto do Tangram.

4.3.1 Configuração e execução de modelos heterogêneos de simulação

Uma vez realizado o trabalho de modelagem de uma federação, o oferecimento de mecanismos de transição para as etapas de configuração e execução é um requisito a ser atendido pelo Tangram. Estas etapas têm como propósito a geração automática dos *gateways* e dos arquivos XML de configuração para os federados. Estas etapas estão implementadas no protótipo atual do DCB.

Os arquivos XML que possuem a configuração de cada federado são derivados de um primeiro arquivo XML que possui a configuração da federação. Este arquivo, gerado pela ferramenta de modelagem, é interpretado pela ferramenta de configuração ao gerar os arquivos individuais dos federados.

Contudo, a geração automática depende do prévio conhecimento, por parte do módulo de configuração, da tecnologia utilizada na construção do federado. Caso isso

não seja possível, um protótipo do *gateway* é gerado, e o projetista precisa interferir no código deste protótipo com o objetivo de ajustá-lo à interface do federado que ele representa. O módulo de configuração pode ser estendido para suportar novas tecnologias através da adição de *templates*. Os mecanismos de configuração e execução do DCB implementados no protótipo atual são abordados no Capítulo 5.

4.4 Verificação e validação

Na construção de modelos heterogêneos de simulação, assim como na construção de sistemas com outros propósitos, é bastante desejável que as questões relacionadas à verificação e validação sejam consideradas. O termo verificação é utilizado para designar a busca pela correção da implementação de um modelo de co-simulação de acordo com sua especificação. O termo validação é utilizado para designar a busca pela fidelidade de um modelo perante o sistema real que ele representa [KIR94]. O nível de detalhes implementado por um modelo interfere na sua fidelidade em relação ao sistema real que ele representa.

Além disso, instrumentos de validação podem ser utilizados focalizando dois objetivos gerais: validações funcional e de desempenho. Na validação da funcionalidade, a meta principal é a correção dos aspectos de cooperação e de comportamento interno de sistemas, de acordo com os objetivos do projetista em relação ao sistema real. Para isso, não é necessário que o desempenho dos modelos de representação seja equivalente ao sistema real, basta que sejam mantidos níveis mínimos para seu correto exercício. A definição de um 'nível mínimo' depende basicamente do tipo de sistema que esteja sendo simulado. Já em atividades de validação que envolvem aspectos de desempenho, o modelo precisa ser fiel ao sistema real também nas características temporais, além da parte funcional [LEE2002].

Na construção de modelos através da composição de partes independentes, acentua-se a importância tanto da verificação bem como da validação. Principalmente, em função do nível de desconhecimento dos detalhes internos do comportamento de cada parte utilizada na construção de um modelo [PAG2000].

Sistemas que incorporam características tais como distribuição, tamanho acentuado, e heterogeneidade, geralmente atingem graus de complexidade mais elevados, dificultando o trabalho de verificação e validação. Por isso, o oferecimento de recursos de software (ferramentas) para apoiar o trabalho de verificação e validação pode trazer benefícios.

Levando em consideração os conceitos de federação e de federado utilizados no DCB e previstos para o Tangram, as atividades de verificação de modelos de co-simulação são particularmente importantes na análise das relações funcionais entre federados. Por exemplo, cada federado possui um conjunto de atributos de entrada e um conjunto de atributos de saída. Cada atributo de saída pode ser a origem de um ou mais atributos de entrada de federados destino. Nas ligações entre atributos erros podem ser evitados através de políticas de verificação. Por exemplo, pode ser feita uma verificação de seqüências indesejáveis de eventos através de um mecanismo automático que compare uma seqüência de eventos gerada por uma execução do modelo com uma seqüência de eventos equivalente porém já existente e considerada correta pelo projetista. Esta seqüência correta pode ser resgatada do sistema real que o modelo

representa, ou ser estabelecida pelo projetista como referência do comportamento desejado para o modelo.

As chances de ocorrência de ligações inconsistentes (ou indesejáveis diante dos propósitos de um modelo) entre atributos aumentam proporcionalmente ao número de ligações. No estabelecimento dessas relações funcionais (ligações) entre federados, a disponibilidade de um software de apoio pode facilitar grandemente o trabalho de verificação. Este trabalho torna-se mais complexo quando envolvidas as partes de gerenciamento de propriedade sobre atributos e sincronização, entre outras.

Nesse contexto, uma ferramenta de apoio à verificação tem como requisito o provimento de funções para a busca de erros em um modelo. São exemplos dessas funções:

- Verificar se existem variáveis de saída configuradas como variáveis de entrada (nas ligações entre federados);
- Verificar se a evolução do tempo em cada federado de um modelo está de acordo com o seu modo de execução. Por exemplo, identificar se um federado configurado como síncrono está equivocadamente utilizando o método do DCB que retorna o GVT assíncrono;
- Identificar os atributos (de federados) não referenciados na configuração dos embaixadores;
- Verificar se as regras que regem a troca de federado proprietário de cada atributo estão consistentes com a configuração das ligações entre atributos.

A especificação e implementação destas funcionalidades num ambiente de modelagem e execução requer a continuidade deste trabalho direcionando os esforços para a construção de ferramentas de apoio à modelagem, fase anterior à configuração e execução (explorada pelo DCB).

Em relação à validação, é desejável que um projetista tenha à disposição informações que facilitem a comparação de um modelo com o sistema real que ele representa. Estas informações podem ser obtidas a partir dos mesmos mecanismos utilizados na verificação.

Quanto ao esquema de funcionamento de um software de apoio à verificação e validação, alguns princípios podem ser assumidos. Inicialmente, um software deste tipo deve manter uma política de comunicação com os demais módulos do ambiente de projeto: módulo de projeto, módulo de configuração, módulo de execução e módulo de tratamento dos resultados do exercício de modelos. No uso destes módulos é conveniente que o projetista possa ‘configurar’ um nível de varredura das informações consideradas relevantes. É importante que as informações a serem monitoradas, tais como atributos de entrada e saída, sejam previamente definidas/selecionadas pelo projetista. O monitoramento dos valores dos atributos segue o conceito de depuração.

4.5 Da concepção à execução de federações heterogêneas

Esta seção tem como objetivo geral discutir um conjunto de fases que envolvem a concepção, construção e execução de modelos de co-simulação à luz dos requisitos do ambiente Tangram. A existência de estágios com objetivos gerais progressivos no desenvolvimento de modelos de representação tende a contribuir para a obtenção de bons resultados. As vantagens podem ser mais significativas na medida em que os modelos tornam-se mais complexos devido à heterogeneidade, tamanho, entre outros fatores. Por isso, justificam-se discussões sobre o uso de estágios de desenvolvimento de modelos que estejam em sintonia com as características do ambiente proposto neste trabalho.

Embora metodologias de modelagem baseadas em orientação a objetos [COP97] sejam vistas como mais adequadas que metodologias tradicionais (estruturadas), esta discussão não é restritiva a nenhuma delas. Ao contrário, é defendida a idéia de manter a apresentação dos estágios num patamar de abstração alto o suficiente que permita contornar restrições quanto à liberdade de projetistas no uso de ferramentas, metodologias, ou técnicas de construção de modelos.

O aumento do número de níveis de abstração também vem sendo visto como uma solução promissora para o problema de queda de produtividade no processo de projeto de sistemas. Esta alternativa tem motivado iniciativas para para que um maior número de decisões de projeto sejam tomadas em níveis de abstração ainda livres de limitações relacionadas a técnicas ou ferramentas. Além disso, torna-se mais fácil definir e limitar as ações a serem desenvolvidas em cada nível visando simplificar e otimizar atividades de especificação e projeto, principalmente se descentralizadas [GER2002].

A seguir são sugeridos, e brevemente comentados, quatro estágios gerais para o desenvolvimento de modelos de co-simulação. Os estágios são: Concepção; Especificação Estática; Comportamento; e Execução. Alguns requisitos precisam ser observados em todos os estágios. Por exemplo, o projetista precisa especificar pré e pós condições de funcionamento para os federados. Esta especificação precisa ser revista a cada estágio, reduzindo assim o risco de adoção/escolha de um federado que não atenda às necessidades de uma determinada parte de um modelo em construção.

4.5.1 Concepção

O estágio de concepção envolve atividades tais como: estudo do sistema existente (ou novo sistema) para o qual se quer construir um modelo; identificação das partes relevantes (processo de abstração) que interferem no problema em estudo; divisão do sistema em módulos que poderão ser representados por federados distintos (particionamento [FER95]); identificação e descrição de propriedades desejáveis aos federados; entre outras.

Particularmente, a descrição das propriedades desejáveis, de acordo com os propósitos de cada parte do modelo (previamente definidas segundo o projetista), é a atividade básica que orienta a busca e seleção/construção dos federados no próximo estágio. Ou seja, a procura/implementação de federados cujas propriedades sejam

suficientemente semelhantes às propriedades descritas para cada uma das partes identificadas por um projetista.

O projetista possui total liberdade e também responsabilidade sobre as decisões tomadas na tarefa de concepção de um modelo de co-simulação, inclusive quanto ao nível de complexidade das partes resultantes da sub-divisão (particionamento) de um modelo de representação. A divisão de um modelo em partes que possam ser representadas individualmente (conceito de componentes) está sujeita a dois fatores maiores que precisam ser considerados:

- Complexidade exagerada: o aumento da complexidade de uma parte dificulta proporcionalmente o trabalho de busca por um federado cujas propriedades possam suprir suas exigências. Contudo, tende a representar da forma mais fiel o sistema real na medida em que, em eventos internos, um federado não sofre interferências de problemas gerados pela necessidade de cooperação;
- Simplicidade exagerada: a redução da complexidade facilita proporcionalmente o trabalho de busca por um federado cujas propriedades possam suprir suas exigências. Contudo, tende a representar de forma menos fiel o sistema real e a aumentar o número de troca de mensagens na medida em que aumenta o número de federados.

4.5.2 Especificação dos federados e interface

A partir do trabalho realizado no estágio anterior (concepção), este estágio tem como propósito principal agregar à federação em construção os federados (componentes) adequados à representação de cada um dos módulos definidos no estágio anterior, seja pela busca ou pela implementação desses federados. A busca pode ser realizada basicamente pela comparação entre as propriedades dos federados disponíveis e as propriedades definidas como desejáveis para cada uma das partes em que o modelo foi dividido pelo projetista.

São exemplos de propriedades: descrição do comportamento interno; tecnologia de implementação; e se está sob restrições de propriedade intelectual. Além disso, neste estágio também são definidos (organizados/estruturados/...) os atributos que compõem a interface de cada federado, mas não sua relação com os atributos dos demais federados, por isso a definição de especificação de interface para este estágio.

Na atividade de busca de federados para representar cada uma das partes previamente definidas, podem ser basicamente utilizadas três diferentes estratégias. A primeira faz uso do módulo de busca e catalogação de federados, como apresentado na seção 4.2. Esta é a estratégia mais atrativa devido à agilidade e maior disponibilidade de informações sobre o federado, já que esse módulo prevê a manutenção de um repositório de federados contendo informações a respeito de suas propriedades (interface, descrição do comportamento interno, entre outras).

Após a busca e seleção de um federado adequado para cada uma das partes de um modelo em construção, tem início a atividade de organização dos atributos desses federados. Nela, além da definição dos atributos, o projetista define as informações a serem utilizadas na configuração dos embaixadores de cada federado (na perspectiva do

DCB). O Capítulo 3 desta tese apresenta o conceito e os propósitos do uso de embaixadores, e o Capítulo 5 apresenta a estrutura interna desses embaixadores.

Na construção de modelos de co-simulação baseada na composição de federados heterogêneos existentes (ou construídos em separado pelo projetista quando necessário), a principal preocupação do projetista é a correta cooperação entre federados cujo controle sobre o comportamento interno pertence aos seus desenvolvedores. Neste estágio tem início a atividade de verificação e validação do modelo.

4.5.3 Especificação das relações de cooperação entre federados

Após a definição dos federados e das informações que regem sua interface, a especificação do o comportamento do modelo é identificada como próximo estágio (configuração dos embaixadores na perspectiva do DCB). Ou seja, origem e destino para as mensagens de entrada e de saída dos federados são descritos.

Basicamente, a configuração do comportamento do modelo é realizada através do estabelecimento de uma ligação dos atributos de saída de cada federado com um ou mais atributos de entrada de outros federados de um modelo (federação). Desta forma, o projetista pode definir as 'relações funcionais' que gerenciam a cooperação entre federados.

Durante a configuração do comportamento o projetista também precisa estar atento para aspectos de validação e verificação do modelo. Justifica-se esta necessidade principalmente devido ao fato de que no estabelecimento de ligações entre atributos podem surgir situações de incompatibilidade entre federados. Se tais situações não puderem ser resolvidas via configuração do comportamento, a substituição do federado por um equivalente pode ser necessária (retorno à concepção). Contudo, o trabalho maior de validação e verificação ocorre quando o modelo é submetido a testes de execução.

4.5.4 Execução

O estágio de execução de modelos de co-simulação engloba a preparação do ambiente necessário ao suporte para a execução do modelo em desenvolvimento, as atividades de validação e a verificação, e a geração de resultados mediante o exercício da co-simulação (execução). Estas atividades geram informações de retorno ao projetista que, por sua vez, pode interferir no modelo com o objetivo de corrigir erros e/ou melhorar a fidelidade em relação ao sistema real.

Como um modelo de co-simulação distribuído tende a incorporar maior complexidade em virtude da heterogeneidade e da distribuição, na preparação do ambiente de suporte à execução de um modelo é relevante que medidas preventivas sejam tomadas antes do disparo da co-simulação. Por exemplo: verificar se existe um processo de cada federado do modelo já em execução no nodo correto; verificar se os embaixadores de cada federado estão também ativos (em execução) nos respectivos nodos dos federados aos quais estão alocados. Em resumo, verificar se o modelo oferece condições para que uma co-simulação seja disparada. A complexidade desta tarefa aumenta proporcionalmente ao nível de heterogeneidade entre os federados e à distribuição geográfica destes federados.

Por fim, a coleta e agrupamento dos resultados de simulação precisam ser tratados durante e ao final da execução de uma co-simulação. A princípio, todos os resultados de uma co-simulação devem ser enviados ao posto de trabalho a partir do qual a execução foi disparada. No entanto, não é objetivo deste ambiente executar atividades de avaliação de desempenho sobre os resultados coletados.

5 PROTÓTIPO DO TANGRAM/DCB

5.1 Introdução

Este capítulo discute o desenvolvimento do protótipo do Tangram/DCB concebido no contexto desta tese. O protótipo incorpora os itens essenciais definidos na especificação do DCB para a configuração e execução de federações heterogêneas. Não são contemplados neste protótipo o suporte para federados assíncronos e o gerenciamento de propriedade.

Num primeiro momento da apresentação do protótipo são discutidas as ferramentas de modelagem e de configuração de federações. Em seguida, o módulo de suporte à execução distribuída é apresentado. Na abordagem desta segunda parte, o g*ateway* é descrito a partir de seu código original, pois é a única parte do DCB que sofre alterações/modificações na implementação de acordo com a interface do federado com o qual deve interagir.

As demais partes do módulo de suporte à execução sofrem apenas ações de configuração, sem alteração de código. Assim, estas partes são discutidas a partir de diagramas de estado UML (*Unified Modeling Language*) [FOW2000] em nível de especificação.

A parte do protótipo que incorpora a comunicação entre atributos de saída/entrada de federados envolvidos em uma co-simulação, a geração automática de *gateways* e a configuração de embaixadores através de arquivos XML [STY2001] foi implementada em cooperação com um trabalho em nível de mestrado no PPGC-UFRGS [SPE2003]. Está em andamento um segundo trabalho em nível de mestrado [SOU2003] cujas atividades envolvem a especificação e implementação dos módulos que deverão compor o ambiente de modelagem do Tangram. A implementação dos módulos relacionados ao gerenciamento de sincronização foi realizada no contexto deste trabalho de tese.

Os protótipos da ferramenta de configuração e dos módulos do DCB foram implementados na linguagem Java. As características para o tratamento de tarefas distribuídas no escopo das pesquisas em simulação distribuída, entre outras comentadas na Seção 2.2.4 desta tese, incentivaram o uso desta linguagem na implementação dos protótipos.

5.2 Modelagem e configuração de federações

Conforme a definição do Tangram, apresentada no Capítulo 4, é desejável que as atividades de projeto e implementação de modelos heterogêneos sejam desenvolvidas o mais rapidamente possível. A existência de um módulo que ofereça uma área de trabalho voltada para o projetista de modelos contribui para agilizar estas atividades.

Deste modo, o protótipo atual do Tangram/DCB prevê a construção de um módulo de modelagem voltado para a etapa de projeto de federações heterogêneas. Os requisitos para este módulo são discutidos no Capítulo 4 desta tese.

Para a tarefa de configuração de uma federação com vistas a sua execução sobre o DCB, o protótipo também oferece uma ferramenta de configuração. Basicamente, esta ferramenta utiliza um arquivo XML de saída da ferramenta de modelagem, que contém informações necessárias sobre o modelo, para a criação do *gateway* e de um outro arquivo XML privado para cada federado.

5.2.1 O módulo de modelagem

Embora discutido nesta tese em termos de requisitos (Capítulo 4), está em andamento um trabalho em nível de mestrado [SOU2003] cujo objetivo é a especificação e implementação de um ambiente de modelagem de federações à luz da arquitetura do DCB. Atualmente existe uma implementação parcial do ambiente que inclui a interface gráfica para modelagem e o repositório de componentes e permite gerar um arquivo XML com a configuração de federações. Este arquivo é utilizado pela ferramenta de configuração (abordada na próxima seção) do DCB para geração dos *gateways* e dos arquivos XML individuais de cada federado.

Este módulo possui quatro partes principais: uma ferramenta gráfica de modelagem; um repositório de componentes; um assistente para adaptação de interface; e um assistente de importação.

Ao armazenar um federado em um repositório para que possa ser utilizado na construção de um modelo com o uso da ferramenta de modelagem, o projetista precisa declarar explicitamente sua interface através de um assistente para adaptação de interface. Deste modo, não é necessário o conhecimento de detalhes internos do federado quando ele for integrado a uma federação.

Na declaração da interface dos componentes, para fins de modelagem, são definidos pontos de acesso. Cada ponto de acesso pode ser definido em diferentes níveis de abstração. Por exemplo, como um conjunto de portas em nível RT, ou como métodos que utilizam parâmetros de entrada e saída para níveis mais altos de abstração.

Como resultado do trabalho de modelagem é gerada uma descrição XML do modelo (federação). Através desta descrição a ferramenta de configuração tem acesso às informações dos federados necessárias à configuração e execução da federação.

A Figura 5.1 mostra um exemplo de arquivo XML com informações de dois dos federados da federação utilizada como estudo de caso nos experimentos apresentados no Capítulo 6. Nesta figura, a linha 3 informa a identificação da federação, a linha 5 informa a identificação do primeiro federado (federado Display no modelo do SistemaGPSAlerta), o nodo em que o federado está localizado (para comunicação

remota), o tipo de sincronização do federado e a localização física do federado no nodo. Informações equivalentes são apresentadas na linha 14 para o federado com id = “3”.

```
01<CONFIG>
02
03<FEDERATION id="1">
04
05<FEDERATE id="1" ip="127.0.0.1" port="4000" type="synchronous"
       code_location="C:\Java\dcb\gpsalerta_driver\display" interface_type="java">
06    <ACCESSPOINT>
07      <METHOD uid="1.0" name="setOutputValue" return_type="void">
08        <PARAMETER uid="1.1" name="value" type="int"/>
09        <PARAMETER uid="1.2" name="channel" type="int"/>
10      </METHOD>
11    </ACCESSPOINT>
12</FEDERATE>
13
14<FEDERATE id="3" ip="127.0.0.1" port="4002" type="synchronous"
       code_location=
              "C:\Java\dcb\gpsalerta_driver\gpsalerta" interface_type="java">
15    <ACCESSPOINT >
16      <METHOD uid="1.0" name="atendeTeclado" return_type="void">
17          <PARAMETER uid="1.1" name="key_pressed" type="int"/>
18      </METHOD>
19    </ACCESSPOINT>
20    <ACCESSPOINT>
21      <PORT uid="3.0" name="value" port_type="OUT" type="int">
22           <DESTINATION federationid="1" federateid="6" attribute="3.2"/>
23      </PORT>
24      <PORT uid="4.0" name="channel" port_type="OUT" type="int">
25           <DESTINATION federationid="1" federateid="6" attribute="3.1"/>
26      </PORT>
27    </ACCESSPOINT>
28    <ACCESSPOINT>
29      <METHOD uid="7.0" name="tf0Method" return_type="void">
30        <PARAMETER uid="7.1" name="Coord" type="String"/>
31      </METHOD>
32    </ACCESSPOINT>
33</FEDERATE>
34
35  ... (demais federados)
36</FEDERATION>
37
38</CONFIG>
```

Figura 5.1: Arquivo XML de configuração de uma federação

Para a interface de um federado podem ser definidos múltiplos pontos de acesso. Cada ponto de acesso pode possuir definições de interface para mais de um nível de abstração. No nível RT, por exemplo, o ponto de acesso pode ser definido como um conjunto de portas, cada uma com seu próprio tipo de dado. Em níveis mais altos de abstração, tais como de mensagem ou de serviço, o ponto de acesso pode ser definido como um conjunto de métodos de acesso que utiliza parâmetros de entrada e saída.

Por exemplo, as linhas 6 a 11 da figura informam que o federado possui um método de acesso que deve ser instanciado pelo seu *gateway* cuja chamada requer valores nos parâmetros com identificação ‘1.1’ e ‘1.2’. Eles definem os atributos de entrada deste federado.

Já o federado com id = “3” possui três pontos de acesso, dois deles compostos por um método de entrada cada (linhas 15 a 9 e 28 a 32) e um deles composto por duas portas de saída tendo como destino o federado com id = “6” (linhas 20 a 27).

A Figura 5.2 mostra um exemplo da área de trabalho da ferramenta de modelagem onde é apresentada uma visualização gráfica dos componentes na construção de um modelo exemplo. Este modelo exemplo é utilizado como parte do estudo de caso apresentado no próximo capítulo. Na etapa de modelagem representada nesta figura apenas a ligação entre os pontos de acesso dos componentes são mostrados. Os detalhes de interface permanecem ocultos.

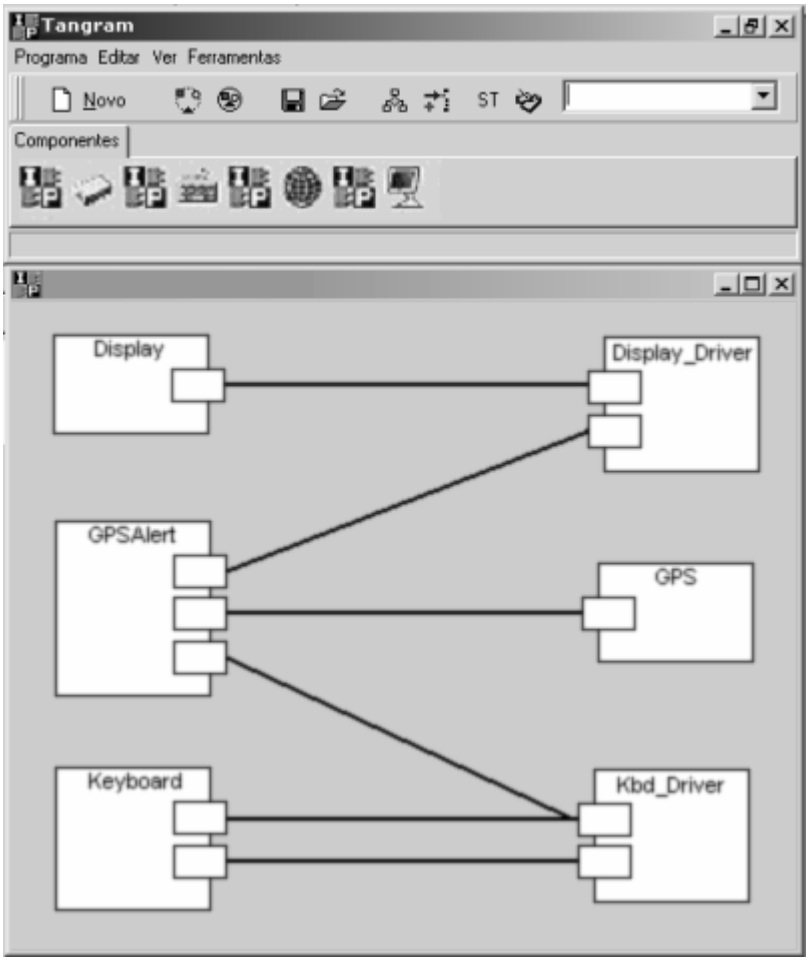

Figura 5.2: Conexão de componentes através de pontos de acesso

Num segundo momento, as portas e métodos de interface contidos nos pontos de acesso são visualizados. A Figura 5.3 ilustra a área de trabalho apresentando as portas e métodos. Por exemplo, o componente GPS Alerta possui as portas de saída *channel* e *value* cujos valores são enviados ao método *SendSignal* do federado Driver_Display.

Na ferramenta de configuração (próxima seção) a construção de *gateways* para os componentes pode ser guiada por moldes (*templates*) genéricos. Por exemplo, um molde para federados descritos em 'C' pode ser usado para integrar quaisquer federados 'C'. Se o uso de um determinado molde não é suficiente por si só para integrar um componente, intervenções no seu código podem ser feitas pelo projetista. O ambiente permite a criação de novos moldes além dos atuais. Os moldes existentes são: Java, *Sockets*, e JNI (*Java Native Interface*) para federados C/C++ com uso ou de biblioteca de ligação dinâmica ou de chamada direta a uma JVM (*Java Virtual Machine*). A especificação, definição e construção da solução baseada em moldes são contribuições da dissertação de mestrado [SPE2003].

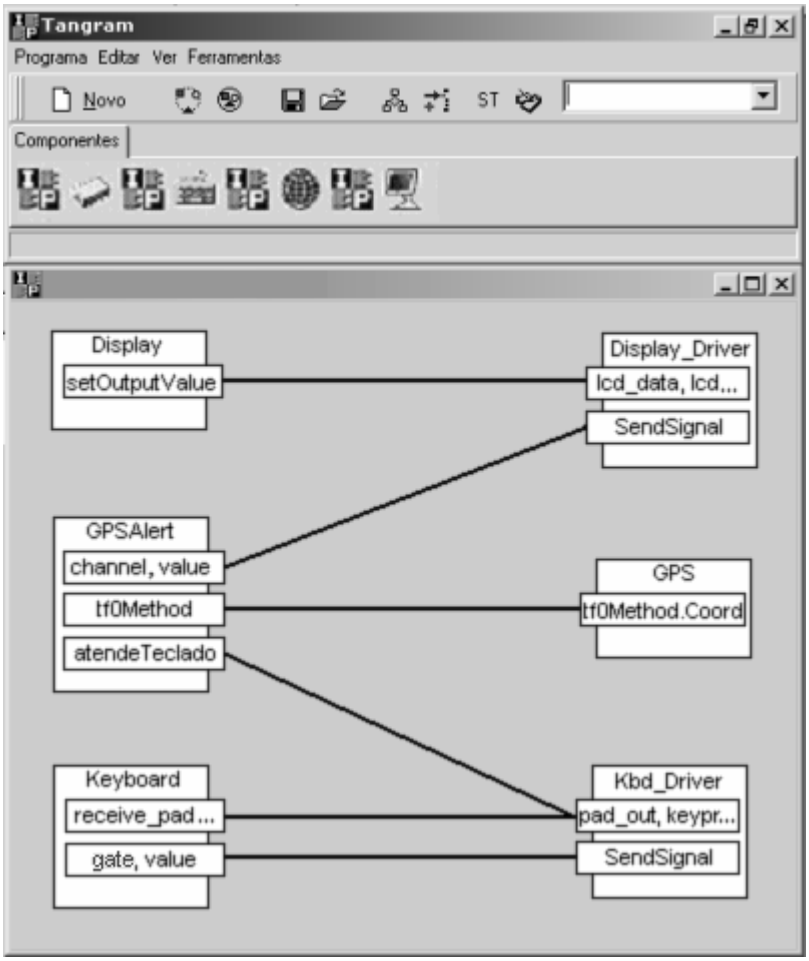

Figura 5.3: Conexão de componentes com visualização de detalhes de interface

É importante ressaltar que a inexistência de um molde não impede a integração de um componente cuja tecnologia não esteja prevista no ambiente. Os *gateways* podem ser criados diretamente pelo projetista. Embora neste caso exista o trabalho de codificação, a tendência é que ele seja expressivamente menor se comparado a alterações na ferramenta de suporte à co-simulação, ou mesmo no componente.

Em situações onde as interfaces de componentes interconectados não combinam, uma adaptação pode ser necessária. Para isso, as ligações incompatíveis (sinalizadas na área de trabalho da ferramenta de modelagem) podem ser diretamente adaptadas se o código do componente for disponível, ou através da construção de *wrappers* quando o código dos componentes estiver sob restrições de acesso (propriedade intelectual). Para isso, o módulo de modelagem prevê a existência de um assistente de adaptação (parte do trabalho em andamento) que orienta a adaptação dos componentes, ou automatiza a geração de *wrappers* [GHA2002], com base na descrição dos componentes mantida pelo módulo que gerencia os repositórios.

5.2.2 A ferramenta de configuração

A tarefa de configuração pode ser vista como o conjunto de passos que prepara uma federação, resultante de um prévio trabalho de modelagem, para a execução sobre o DCB. As informações utilizadas por esta ferramenta são fornecidas através de um arquivo XML gerado automaticamente pelo módulo de modelagem. Para cada um dos federados este arquivo contém informações sobre a identificação de atributos de entrada e saída, tipo de sincronização, nodo em que o federado deve ser executado, identificação do federado e da federação.

A ferramenta de configuração possui duas tarefas principais: a geração dos *gateway*s e a geração dos arquivos XML individuais de cada federado. Inicialmente, o código do *gateway* de cada um dos federados é gerado. Nesta tarefa podem ser utilizados os moldes para a geração de *gateways* para federados descritos em diferentes linguagens/tecnologias.

Também a partir do arquivo XML, a ferramenta de configuração identifica as informações pertinentes a cada federado e cria um arquivo XML específico para cada um deles. No início da execução da federação estes arquivos são lidos dinamicamente pelos embaixadores. Deste modo, a não ser pelo *gateway*, os demais módulos do DCB não sofrem nenhuma alteração de código, seja em função da mudança das características de federados em um federação ou do seu uso em diferentes federações Isto evita a necessidade de recompilação do código do DCB.

```
01<?xml version="1.0" encoding="UTF-8"?>
02
03<CONFIG>
04  <INFO federateid="1" federationid="1" localport="4000" type="notime">
05    <ATTRIBUTE id="7.0" name="tf0Method.Coord" type="String">
06      <DESTINATION federationid="1" federateid="3" attribute="7.1" />
07    </ATTRIBUTE>
08    <ATTRIBUTE id="444.2" name="port" type="String">
09      <DESTINATION federationid="1" federateid="444" attribute="444.2" />
10    </ATTRIBUTE>
11    <ATTRIBUTE id="444.1" name="port" type="String" />
12    <ATTRIBUTE id="1.1" name="address.pos" type="int" />
13    <ATTRIBUTE id="2.0" name="clean" type="function" />
14    <ATTRIBUTE id="3.1" name="writeInt.value" type="int" />
15    <ATTRIBUTE id="4.1" name="writeChar.asciiCode" type="int" />
16    <ATTRIBUTE id="5.0" name="initialize" type="function" />
17  </INFO>
18  <FEDERATION id="1">
19    <FEDERATE id="3" ip="local" port="4002" />
20    <FEDERATE id="444" ip="127.0.0.1" port="4444" />
21  </FEDERATION>
22</CONFIG>
```

Figura 5.4: Arquivo XML de configuração de um federado

A Figura 5.4 apresenta um exemplo de arquivo XML gerado pela ferramenta de configuração para um federado de identificação ‘1’ e tipo de sincronização configurado como ‘*notime’*. Este federado possui os seguintes atributos de saída:

- Identificação do atributo de saída: id = “7.0”; identificação do respectivo atributo de entrada no federado destino: id = “7.1”; identificação do federado destino: federateid = “3” (linhas 5, 6 e 7 da Figura 5.4);
- Identificação do atributo de saída: id = “444.2”; identificação do respectivo atributo de entrada no federado destino: id = “444.2”; identificação do federado destino: federateid = “444” (linhas 8, 9 e 10 da Figura 5.4). A identificação não pode ser repetida em atributos de um mesmo federado.

Este federado possui os seguintes atributos de entrada (linhas 11 a 16 da Figura 5.4):

- Atributo de entrada: id = “444.1”;
- Atributo de entrada: id = “1.1”;
- Atributo de entrada: id = “2.0”;
- Atributo de entrada: id = “3.1”;
- Atributo de entrada: id = “4.1”;
- Atributo de entrada: id = “5.0”.

As linhas 19 e 20 da Figura 5.4 indicam, para cada federado destino, o endereço IP (*Internet Protocol*) em que o federado está executando e a respectiva porta de comunicação utilizada.

- Federado com id="3" está executando no nodo com ip="local" e utilizando a porta "4002";
- Federado com id="444" está executando no nodo com ip="127.0.0.1" e utilizando a porta "4444".

Além da configuração da federação, a especificação XML é utilizada para determinar se federados destino são locais ou remotos. Para federados locais, a troca de mensagens é realizada através da chamada direta de métodos que não utilizam serviços de rede. Já os federados remotos dependem de serviços de rede (*sockets*). A diferenciação no mecanismo utilizado para troca de mensagens entre federados remotos e locais contribui, principalmente, com a otimização de desempenho. No exemplo da Figura 5.4, o federado destino está no mesmo nodo do federado origem, pois a tag ‘ip’ possui o valor ‘local’. Se o federado estiver em um nodo remoto, ‘ip’ recebe o número IP deste nodo.

5.3 Suporte à execução distribuída de federações

O mecanismo de suporte à execução distribuída de federações (para cada federado) é composto pelo *gateway*, pelos embaixadores, e pelo NDCB. A Figura 5.5 apresenta a estrutura interna de cada um destes módulos de acordo com o protótipo implementado.

Nesta figura, em comparação com a arquitetura do DCB apresentada no Capítulo 3, observa-se que o protótipo apresenta o *gateway* dividido em dois módulos e a existência de um módulo adicional denominado *DCBMThread*. Embora o protótipo apresente estes módulos adicionais, a fidelidade com a estrutura conceitual do DCB é mantida.

A inclusão do *DCBMthread* tem o objetivo de permitir que múltiplos federados de um mesmo nodo sejam executados sobre uma mesma JVM. Esta é uma decisão de implementação. Ou seja, considerando que cada federado possui seu *gateway*, seus embaixadores e NDCB particulares, o DCB de cada federado é executado em uma *thread* distinta sobre uma mesma JVM, ao invés do uso de uma execução individual de uma JVM para cada federado.

São dois os benefícios diretos desta decisão de implementação. Primeiro, a exigência de memória é bastante reduzida com a execução de uma única JVM. Em segundo, a troca de mensagens entre federados locais pode ser feita pela chamada direta de métodos de comunicação disponíveis em *DCBMThread*. Deste modo, o uso de serviços de rede se dá apenas na comunicação com federados remotos.

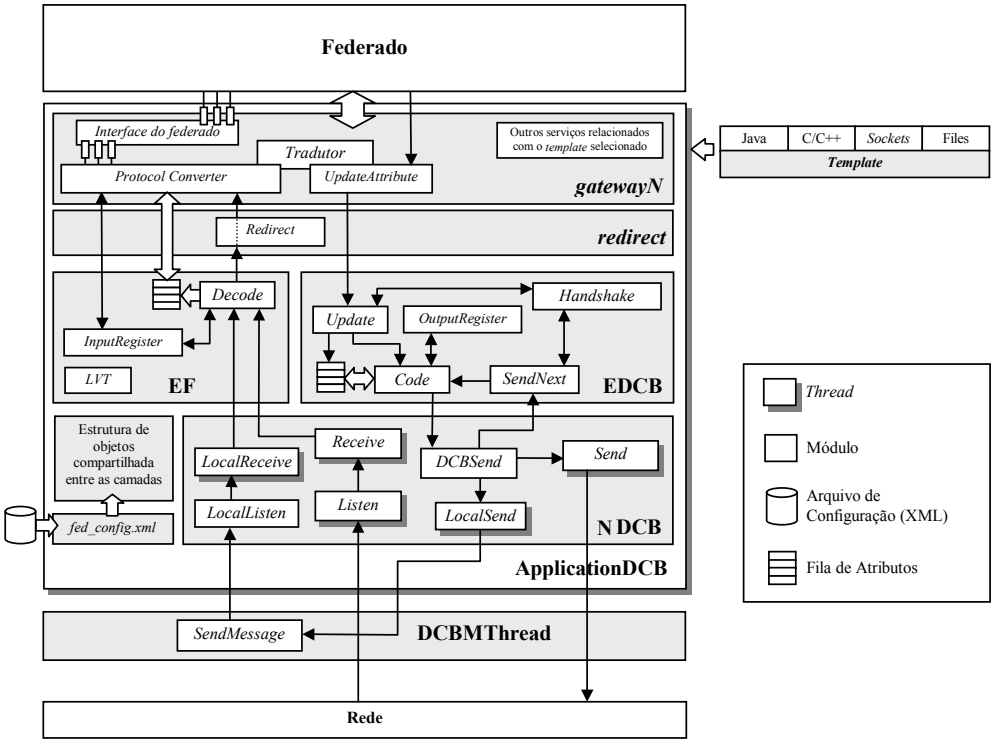

Figura 5.5: Protótipo do DCB

A composição geral do DCB, representada na Figura 5.5, identifica os módulos do g*ateway*, do EF, do EDCB e do NDCB. O g*ateway* tem a responsabilidade de manter os serviços de interface com o federado com propósitos de atualização de atributos de qualquer natureza (entrada, saída, tempo). Na Figura 5.5, o g*ateway* está dividido em dois sub-módulos, um que mantém o nome de *redirect,* e outro denominado *gatewayN***,** onde o *N* significa a identificação do federado que ele representa. Assim, se por exemplo um federado de identificação '1' recebe uma mensagem, o módulo *redirect* direciona esta mensagem para o *gateway1*, módulo que possui a implementação do tratamento de interface do federado. O módulo *redirect* tem como objetivo apenas o direcionamento da mensagem para o *gatewayN* do federado destino.

A divisão do *gateway* em dois módulos também é uma solução de implementação do protótipo para que mútiplos federados possam ser executados em uma mesma JVM. Maiores detalhes desta solução para o *gateway* são apresentados na próxima seção deste capítulo.

O g*atewayN* mantém troca de dados tanto com o EF quanto com o EDCB. A comunicação entre o g*atewayN* e o EDCB ocorre com o propósito de repassar ao EDCB requisições de envio de mensagens. Para isso, o EDCB oferece ao g*atewayN* uma primitiva local de envio, representada na Figura 5.5 pelo módulo denominado *Update*. O EDCB então solicita ao NDCB o uso de serviços de rede para o envio da mensagem ao nodo do federado destino, ou serviços locais de comunicação se o federado destino estiver no mesmo nodo. Esta solicitação é realizada com o uso do módulo *DCBSend* da Figura 5.5.

O caminho inverso (recebimento de mensagens) é similar. Quando o núcleo do DCB recebe uma mensagem, ele solicita ao EF o repasse da mensagem recém chegada para o *gatewayN* que representa o federado destino. Este serviço é representado na Figura 5.5 pelos módulos *Receive* e *LocalReceive* no NDCB, e pelo módulo *Decode* do EF. Em seguida, o EF solicita ao *gatewayN* a conversão dos dados para o formato reconhecido pelo federado destino e a sua entrega (módulo *Protocol Converter*).

Os módulos EF e EDCB cooperam, independentemente dos serviços de comunicação, essencialmente para a execução de atividades de sincronização do tempo de simulação e de gerenciamento de propriedades (esta última ainda não implementada no protótipo).

As seções a seguir apresentam cada um dos módulos do protótipo. Na apresentação do *gateway* e do *gatewayN*, excepcionalmente, são utilizados trechos de código com o propósito de melhor apresentar/identificar as transformações necessárias no seu conteúdo para torná-lo apto a reconhecer a interface de diferentes federados. Esta característica é particular para este caso. Os demais módulos têm seu código estático para qualquer federado, como descrito no Capítulo 3. Por isso, na apresentação de seu comportamento, são utilizados diagramas de estado UML em nível de especificação.

Diagramas de estado permitem descrever o comportamento de um sistema através da apresentação dos possíveis estados de cada objeto em particular e das mudanças desses estados perante a ocorrência de eventos. A sintaxe das descrições utilizadas nos diagramas é: *evento [condição] /ação*. A ocorrência de um *evento* causa uma transição de estado que ocorre se a *[condição]* for satisfeita, sendo então executada a */ação*. Os elementos desta sintaxe são opcionais nas transições de estado.

5.3.1 Os módulos *redirect* e *gatewayN*

Utilizando o arquivo XML que possui as informações de uma federação, o módulo de configuração tem a tarefa de gerar um *gatewayN* para a interface de cada federado, e um *redirect* para todos os federados que serão executados em uma mesma JVM. O *gatewayN* deve reconhecer todos os atributos de entrada e saída do federado respectivo.

```
1-public class Gateway   // classe que possui o método Redirect
2-{
3-…
4-        public void Start(int gatewayVal)  {
5-                gVal= gatewayVal;
6-                // índice do gateway que representa o federado desejado na comunicação
7-                switch (gVal)  {
8-                        case 1:    { Gateway1.SetPointer(App); break; }
9-                        case 2:    { Gateway2.SetPointer(App); break; }
10-                       case 3:    { ...; break; }
11-               }
12-       }
13-       public void Redirect(int AttributeID)  {
14-       // redireciona a chamada originada do EF para o Gateway  do respectivo federado
15-               switch (gVal) {
16-                       case 1: { Gateway1.ProtocolConverter(AttributeID); break; }
17-                       case 2: { Gateway2.ProtocolConverter(AttributeID); break; }
18-                       case 3: { …;  break; }
19-               }
20-       }
21-}
```

Figura 5.6: Estrutura do código do gateway que realiza redirecionamento

A Figura 5.6 apresenta a estrutura do código da parte do módulo *redirect* (Figura 5.5) que mantém a tarefa de direcionar as solicitações de comunicação, vindas de outros federados, para o *gatewayN* do federado cuja identificação de destino é equivalente à da mensagem. Para isso, o *gatewayN* utiliza o mesmo índice numérico de identificação de cada federado que existe no arquivo XML. Este índice substitui o *N* em *gatewayN*. Por exemplo, para um federado de identificação '2' no XML, a ferramenta de configuração cria uma instância da classe *Gateway2* (linhas 9 e 17 da Figura 5.6). São estas as linhas que sofrem alteração, na classe *Gateway* (linha 1 da figura), para diferentes federações. Assim, a classe *Gateway* utiliza o índice de cada federado para a inicialização do *gatewayN* de cada federado (método *Start* na linha 4 da Figura 5.6) e para o redirecionamento das mensagens (método *Redirect* na linha 13).

A Figura 5.7 apresenta a estrutura do código dos g*atewayN's,* que são dependentes da interface do federado que devem representar. Essencialmente, um g*atewayN* presta serviços de interface para o gerenciamento de tempo e para a troca de mensagens tendo o federado que representa como origem ou destino.

O método *UpdateLVT*, na linha 3 da Figura 5.7, deve ser utilizado por federados temporizados sempre que houver avanço no tempo por iniciativa do próprio federado. Este método utiliza serviços de gerenciamento de tempo do EF, retornando para o federado um valor consistente de LVT.

```
1-public class Gateway1 {
2-…
3-   public static String updateLVT(String federateLVT) // Recebe LVT federado e solicita atualização ao EF
4-      { return (App.NewEF.updateLVT(federateLVT)); }
5-   public static String returnGVT() // monitora o avanço do LVT do federado para não ultrapassar o GVT
6-     { return (App.NewDCB.getGVT());  }
7-   public static void ProtocolConverter(int AttributeID)
8      {
9-       …
10      switch (AttributeID) {
11-          case 1: {
12-             A0 = App.NewEF.getAttributeReceived("1.1");
13-             if(A0 != null)
14-             {          controlaB(A0.Value, A0.timestamp);
15-                        App.NewEF.AttributeRemove(A0); // remove msg da lista de entrada mantida pelo EF
16-   } break;
17-          case ...
18-      }
19-   }
20-}
21-/////////////// INICIO FUNÇÕES NATIVAS DO FEDERADO ///////////////
22-public static void controlaB(String atividade, String timestamp)
23-  { Fed.controlaB(atividade,timestamp);    }
24-///////////////// FIM FUNÇÕES NATIVAS DO FEDERADO /////////////////
25-//////////////////////////////// INICIO TRADUTOR ////////////////////////////////////////
26-public long ToLong(String value) {
27-   return Long.parseLong(value); }
28-public …
29-…
30-public void UpdateAttribute(String Name, String Value, String timestamp)
31-  { App.NewEDCB.Update(Name,Value,timestamp); }
32-public void UpdateAttribute(String Name, int Value, String timestamp)
33-  { App.NewEDCB.Update(Name,String.valueOf(Value),timestamp);  }
34-public …
35-//////////////////////////////// FIM TRADUTOR ////////////////////////////////////////
```

Figura 5.7: Estrutura do código do gatewayN

Para utilizá-lo, o federado deve fornecer o valor do LVT para o qual deseja evoluir. O EF, então, retorna para o federado um valor igual ou menor ao GVT vigente de acordo com o avanço solicitado. Recebendo este valor, o federado dá continuidade às suas ações internas, podendo requisitar uma nova evolução do LVT, via *UpdateLVT*, a qualquer momento.

O método *UpdateAttribute*, na linha 30, deve ser utilizado por federados para enviar mensagens para um destino via atributos de saída. Este método sofre sobrecarga em função das diferentes combinações possíveis em relação ao tipo dos parâmetros, como exemplificado na linha 32, onde *Value* é tratado como um inteiro.

O método *ProtocolConverter*, linha 7, realiza a identificação do federado e do atributo origem de acordo com a configuração estabelecida ainda na modelagem. Em seguida, executa uma chamada a um segundo método denominado *controlaB* (linha 22) que está preparado para reconhecer o respectivo método nativo do federado. Os métodos nativos de um federado variam de acordo com a interface do federado. Esta ação permite que um valor recebido através de uma mensagem seja repassado ao atributo de entrada previsto do federado.

5.3.2 Embaixadores

A partir dos arquivos XML gerados pela ferramenta de configuração, o DCB tem acesso às informações de configuração do federado. A configuração é a primeira atividade do DCB ao ser inicializado. Uma vez lido o conteúdo dos arquivos XML, a cooperação do federado com o restante da federação se dá através do uso das primitivas disponíveis no *gatewayN,* abordadas na seção anterior. A seguir são apresentados os diagramas de estado do EF, do EDCB e do NDCB.

5.3.2.1 Embaixador do Federado (EF)

O diagrama de estados da Figura 5.8 apresenta o comportamento do EF no suporte a federados do tipo síncrono e *notime*. O EF possui duas finalidades essenciais: oferecer serviços para o recebimento de mensagens provenientes de federados remotos e incorporar primitivas de sincronização. O EF executa apenas atividades locais sobre mensagens de chegada.

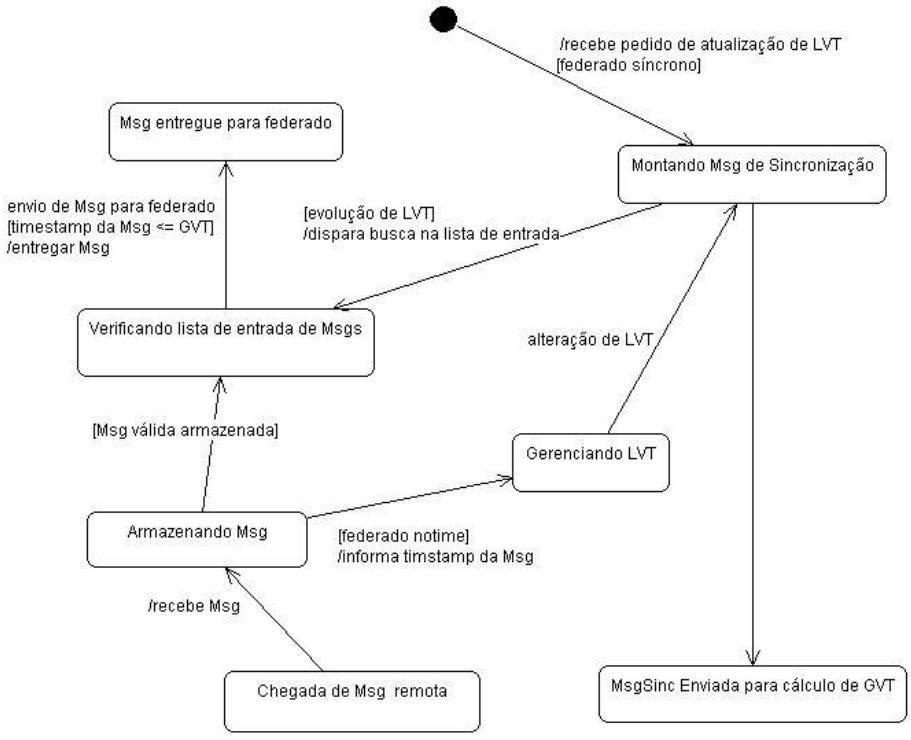

Figura 5.8: Diagrama de estados do EF

Em federados síncronos, o federado é que possui o controle sobre a evolução de seu LVT. Sempre que um federado síncrono executa avanço no tempo, o novo LVT é utilizado na tarefa de sincronização global da federação ('*Montando Msg de Sincronização*' no diagrama da Figura 5.8) e uma busca na lista de mensagens de entrada é disparada ('*Verificando lista de entrada de Msgs*' no diagrama). Esta busca tem como objetivo identificar, de forma ordenada, as mensagens cujo tempo de evento sejam iguais ou menores que o novo LVT e encaminhá-las ao federado ('*Msg entregue para federado*' no diagrama). Uma atualização no LVT também aciona o envio de uma

mensagem com propósitos de atualização do GVT (‘*MsgSinc Enviada para cálculo de GVT*’ no diagrama).

Sempre que ocorre a chegada de uma mensagem originária de outro federado, o EF a inclui na lista de entrada (‘*Armazenando Msg*’ no diagrama). Se o federado destino é do tipo *notime*, o tempo de evento da mensagem recém chegada é informado para um módulo de tratamento de tempo que realiza a atualização de LVTs para federados do tipo *notime* (‘*Gerenciando LVT*’ no diagrama).

5.3.2.2 Embaixador do DCB (EDCB)

O diagrama de estados da Figura 5.9 apresenta o comportamento do EDCB. Este módulo tem como propósito essencial oferecer serviços para o envio de mensagens, tanto de dados como de sincronização, a partir de federados origem. O EDCB executa apenas atividades locais de controle sobre mensagens de saída.

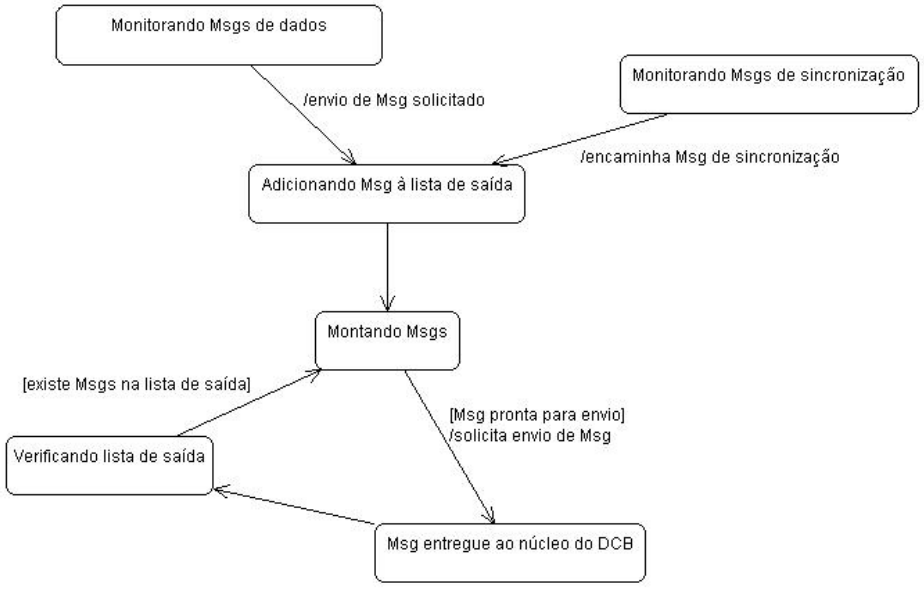

Figura 5.9: Diagrama de estados do EDCB

Para isso, o EDCB monitora a ocorrência de mensagens de dados (‘*Monitorando Msgs de dados*’ no diagrama da Figura 5.9) e de sincronização (‘*Monitorando Msgs de sincronização*’ no diagrama). Uma vez recebida uma mensagem de dados, ela é adicionada a uma lista de mensagens de saída (‘*Adicionando Msg à lista de saída*’ no diagrama).

Depois de armazenada para envio, a mensagem é traduzida para um formato padrão de uso interno do DCB, contudo mantendo as propriedades originais (‘*Montando Msgs*’ no diagrama). No passo seguinte, a mensagem é repassada ao NDCB para que serviços de rede ou de comunicação local sejam utilizados no seu envio.

Em seguida, o EDCB verifica a existência de mais mensagens na lista de saída. Se houverem, executa novamente o procedimento de tradução e repasse de cada mensagem ao NDCB.

Para mensagens de sincronização, o EDCB mantém uma política semelhante (‘*Monitorando Msgs de sincronização*’ no diagrama). Contudo, para este tipo de mensagens, ações de gerenciamento de tempo (internas) são executadas antes que elas

sejam armazenadas na lista de saída. Esta última é comum para ambos os tipos de mensagens.

Para federados *notime*, é o EDCB que mantém o mecanismo que atribui um valor consistente para o campo *timestamp* da mensagem. Embora este valor não seja utilizado pelo federado destino, ele é utilizado pelo DCB no destino para garantir a ordem de entrega das mensagens ao federado. Este mecanismo é apresentado na seção que discute o gerenciamento de tempo neste capítulo.

5.3.3 Núcleo do DCB (NDCB)

O diagrama de estados da Figura 5.10 apresenta o comportamento do NDCB. Ele oferece serviços de envio e recebimento de mensagens de modo complementar às atividades incorporadas pelo EF e pelo EDCB, módulos cujas ações são restritas ao nodo local do federado que representam.

Assim, o NDCB é responsável pelas tarefas de envio de mensagens codificadas pelo EDCB para o destino e pelo recebimento de mensagens oriundas de outros federados, repassando-as para o EF que realiza a tarefa de decodificação. O NDCB não diferencia mensagens de dados e de sincronização.

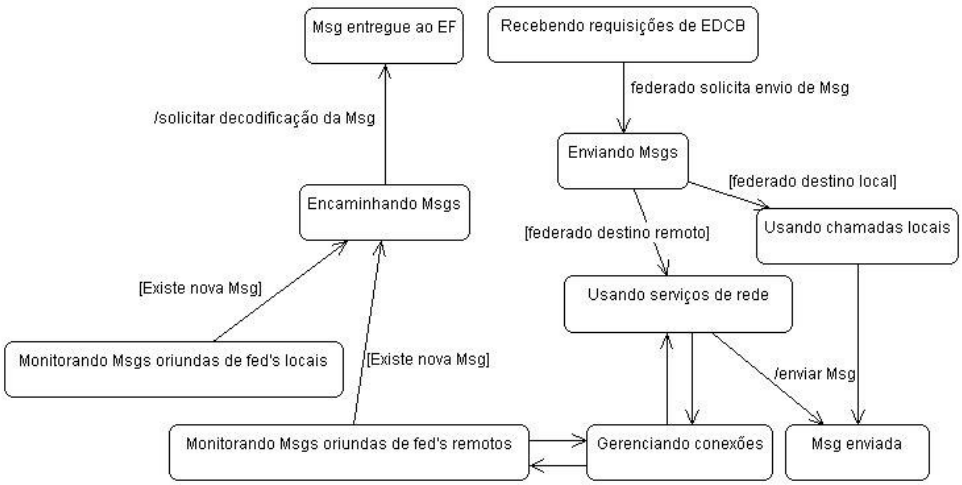

Figura 5.10: Diagrama de estados do núcleo do DCB

No apoio às atividades do EF, o NDCB monitora a chegada de mensagens de outros federados locais (‘*Monitorando mensagens oriundas de fed’s locais’* no diagrama da Figura 5.10) e de federados remotos (‘*Monitorando mensagens oriundas de fed’s remotos’* no diagrama) em módulos de código distintos. Uma vez detectada a chegada de uma nova mensagem, ela é encaminhada ao EF (‘*Encaminhando Msgs*’ no diagrama) para que seja decodificada.

No apoio às atividades do EDCB, o NDCB monitora as solicitações de envio provenientes do EDCB (‘*Recebendo requisições de EDCB*’ no diagrama), executando as operações necessárias para atender às requisições de envio. Para isso, o NDCB verifica se o federado destino é remoto ou local em relação ao nodo origem. Se local, utiliza chamadas diretas a procedimentos locais para envio de mensagens (‘*Usando*

chamadas locais' no diagrama). Se o federado destino é remoto, utiliza serviços de rede para envio de mensagens ('*Usando serviços de rede*' no diagrama).

Para operações remotas de entrada e saída, o NDCB mantém operações de gerenciamento de conexões por *sockets* ('*Gerenciando conexões*' no diagrama). Para recebimento de mensagens, o NDCB pode aceitar 'n' pedidos de conexão de nodos remotos, atribuindo cada conexão a uma *thread* distinta. Para o envio de mensagens, caso não exista uma conexão com o destino pretendido, o NDCB executa um pedido de conexão que se mantém aberta enquanto o federado destino estiver ativo.

5.3.4 Gerenciamento de tempo mantido pelo DCB (GVT/LVT)

Os federados que compõem uma federação processam eventos localmente. Os eventos podem ser gerados pelo próprio federado, ou enviados de outros federados. A ordem em que os eventos são processados é regida pelo tempo de simulação local (LVT), controlado individualmente pelo próprio federado. Contudo, a existência de múltiplos LVT's, um para cada federado, e a possibilidade de solicitar a execução remota de eventos, podem comprometer a ordem global em que eventos são executados. Nestas condições, não há garantias da equivalência entre os valores dos diferentes LVT's existentes.

Para garantir a consistência na ordem global de execução de eventos em um modelo de simulação distribuído (sincronização) [LAM78], a adoção do conceito de tempo virtual global (GVT-*Global Virtual Time*) torna-se essencial [STE95]. Para isso, embora os federados mantenham seus tempos locais, sua evolução deve respeitar os limites impostos pelo GVT, que é atualizado de forma atômica na federação. O avanço do GVT determina o progresso de uma simulação. O Capítulo 3 deste texto discute os modos de atualização do GVT em função do tipo de sincronização (síncrona / assíncrona) no DCB.

Embora o uso do conceito de GVT ofereça suporte suficiente para garantir consistência na sincronização, a sua implementação geralmente não é trivial, podendo provocar um aumento expressivo na troca de mensagens. O Capítulo 2 desta tese aborda algums técnicas para manutenção de tempo global e algumas características desejáveis para soluções com esta finalidade.

À luz dessas técnicas e características, foi criado um novo federado denominado Fedgvt para exercer o tratamento do GVT no protótipo do DCB. Este federado tem como objetivo geral receber o valor dos LVT's dos demais federados, executar o cálculo do GVT, e informar todos os federados do novo GVT se houver avanço. Para isso, os EF's dos demais federados enviam o LVT local para o Fedgvt sempre que houver uma tentativa de avanço de LVT, com ou sem sucesso.

5.3.4.1 O federado Fedgvt

Para gerenciar o GVT o Fedgvt mantém uma lista com os LVT's recebidos dos federados. Esta lista possui a identificação de federação e federado proprietário de cada LVT armazenado. O Fedgvt também mantém um atributo denominado '*lookahead*', cujo valor significa o período de tempo futuro para o qual nenhuma solicitação de execução será realizada (ver Seção 2.2.2). O GVT é dado pelo valor do menor LVT dentre os LVT's mantidos na lista, mais o *lookahead*.

$$GVT = menor(LVT) + lookahead$$

Se o valor de GVT for alterado com a chegada de uma mensagem informando o avanço de um LVT, todos os federados da federação são informados do novo valor do GVT. Na sua atualização, a cada novo LVT que o Fedgvt recebe, são executados basicamente os seguintes passos:

1. O Fedgvt verifica se o LVT recebido é proveniente de um federado novo (que ainda não consta na lista). Se positivo, inclui mais uma entrada na lista de controle de LVT's. Se negativo, atualiza o valor do LVT existente.
2. O Fedgvt executa uma busca na lista para identificar o menor LVT. Se o menor LVT mais o valor do *lookahead* (LVT+*lookahead*) for diferente do GVT atual, então GVT=LVT+*lookahead*. Em seguida todos os federados da federação são informados do novo GVT. Caso contrário, aguarda o recebimento de novos LVT's sem requisitar envio de mensagens.

Ao inicializar uma federação é importante que o Fedgvt seja disparado antes dos demais federados, pois, ao primeiro sinal de evolução no tempo, ele executa a atualização inicial do GVT e a comunica para todos os demais federados. Se um federado qualquer executa antes disso uma operação sobre um atributo de saída, as tentativas de enviar o LVT ao Fedgvt para cálculo de GVT irão falhar.

O mesmo problema ocorre entre federados específicos de um modelo. Ou seja, um federado origem não pode enviar uma mensagem para um federado destino que ainda não tenha sido disparado na federação.

Este é um problema restrito ao escopo da implementação do protótipo do DCB que apenas impõem uma ordem de disparo entre federados, não comprometendo a simulação. No entanto, é desejável que o disparo de federados possa ocorrer sem que o projetista tenha preocupações com a consistência do modelo em relação aos aspectos de sincronização. Para isso podem ser concebidas estratégias mais intuitivas que reduzam a preocupação do projetista em relação à ordem de disparo de federados.

5.3.4.2 Requisitos de configuração do Fedgvt

Para o correto funcionamento do Fedgvt, alguns requisitos de configuração precisam ser observados. Eles interferem basicamente no conteúdo dos arquivos XML de configuração fornecidos pelas ferramentas de modelagem e de geração da federação. A Figura 5.11 demonstra, em destaque, o conteúdo exigido nos arquivos XML de configuração dos federados, exceto para o Fedgvt.

A linhas 5, 6 e 7 da figura indicam que a ocorrência de solicitações para avanço de tempo do federado com identificação '2' (linha 4) devem ser enviadas para o federado '444' (identificação do Fedgvt) através do atributo '444.2'. Este federado está no endereço indicado na linha 10.

O atributo '444.2' é reservado para o DCB manter o gerenciamento de tempo. Ele não pode ser utilizado pelos federados como identificação de atributos de saída ou entrada.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02
03 <CONFIG>
04 <INFO federateid="2" federationid="1" locaPLort="4001" type="synchronous">
05   <ATTRIBUTE id="444.2" name="port" type="String">
06     <DESTINATION federationid="1" federateid="444" attribute="444.2" />
07   </ATTRIBUTE>
08 </INFO>
09 <FEDERATION id="1">
10   <FEDERATE id="444" ip="127.0.0.1" port="4444" />
11 </FEDERATION>
12 </CONFIG>
```

Figura 5.11: Informações de gerenciamento do GVT no XML dos federados

Já a Figura 5.12 demonstra, em destaque, o conteúdo do arquivo de configuração do federado Fedgvt (fedgvt.xml). Nele, o atributo de saída de identificação 444.1 deve estar apontando para todos os demais federados participantes da federação.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02
03 <CONFIG>
04   <INFO federateid="444" federationid="1" locaPLort="4444" type="synchronous">
05     <ATTRIBUTE id="444.1" name="port" type="String">
06       <DESTINATION federationid="1" federateid="<num federado(1)>" attribute="444.1" />
07       <DESTINATION federationid="1" federateid="<num federado(2)>" attribute="444.1" />
08       <DESTINATION federationid="1" federateid="<num federado(n)>" attribute="444.1" />
09     </ATTRIBUTE>
10     <ATTRIBUTE id="444.2" name="calcGVT.remoteTime" type="String" />
11   </INFO>
12   <FEDERATION id="1">
13     <FEDERATE id="<num federado(1)>" ip="<num ip(1)>" port="<num porta(1)>" />
14     <FEDERATE id="<num federado(2)>" ip="<num ip(2)>" port="<num porta(2)>" />
15     <FEDERATE id="<num federado(n)>" ip="<num ip(n)>" port="<num porta(n)>" />
16 </FEDERATION>
17 </CONFIG>
```

Figura 5.12: Informações de gerenciamento do GVT no XML do Fedgvt

As linhas 6, 7 e 8 indicam os federados para os quais o Fedgvt deve comunicar o novo GVT quando seu valor for alterado. As linhas 13, 14 e 15 indicam o endereço desses federados.

Por fim, quando um novo federado é inserido em uma federação o seu LVT inicial deve respeitar o estado atual da simulação. Ou seja, o LVT deve ser inicializado de acordo com o GVT vigente.

5.3.5 Interface entre federado e *gatewayN* no controle do tempo local

A Figura 5.13 apresenta um exemplo de código para a interação entre federado síncrono com comportamento ativo (federado que envia mensagens) e respectivo *gatewayN*. O código em destaque nas linhas 3, 5 e 9 desta figura é obrigatório no código de federados síncronos para reconhecimento do GVT, envio de mensagem e atualização de LVT, respectivamente. Neste exemplo são observados três passos que envolvem o avanço do LVT e o envio de uma mensagem. São eles:

- Na linha 3: o atributo inputLVT (do federado) possui o valor do novo LVT proposto por ele. Se este valor é maior que o GVT, o LVT não pode, neste momento, evoluir como requerido pelo federado. Então, a solicitação de envio de uma mensagem do federado origem para um federado destino não pode ser atendida;
- Na linha 5: uma mensagem é enviada para um atributo destino, contendo um valor e respectivo tempo de evento (tempo de execução no destino), através de uma chamada ao método Gateway.UpdateAttribute(...). Esta mensagem é enviada apenas se o valor do LVT desejado pelo federado não ultrapassar o valor do GVT mantido pelo DCB;
- Na linha 9: o atributo currentLVT (também do federado) recebe o valor do novo LVT de acordo com a política de sincronização do DCB. O valor do novo LVT é dado pelo retorno do método Gateway.updateLVT(...), que é limitado pelo valor do GVT mantido pelo DCB.

```
...
01  //////////INICIO COMUNICAÇÃO COM O GATEWAY////////////////////////////////
02
03  if (inputLVT <= Integer.parseInt(Gateway.returnGVT()))
04  // condição que verifica se o avanço de tempo requerido pelo federado não ultrapassa o GVT
05          Gateway.UpdateAttribute ("1.1",mens, String.valueOf(timestamp));
06          // operação que envia requisição para incrementar buffer
07  else
08          System.out.println("Msg não enviada (LVT > GVT). LVT mantido = ao GVT");
09  currentLVT = Integer.parseInt(Gateway.updateLVT(String.valueOf(inputLVT)));
10 // operação que atualiza LVT local
11
12 //////////FIM COMUNICAÇÃO COM O GATEWAY////////////////////////////////
...
```

Figura 5.13: Exemplo de interface entre gatewayN e federado síncrono

No comportamento passivo de um federado síncrono (federado recebe mensagem), uma mensagem só é repassada ao federado (pelo seu EF) quando 'tempo de evento <= GVT'. Neste caso, o federado tem autonomia para decidir se a atualização do LVT para o valor do tempo de evento da mensagem, repassada ao federado pelo seu EF, será concretizada. O valor do LVT mantido pelos embaixadores acompanha a 'vontade' do federado. Ou seja, o LVT interno do federado e o LVT mantido pelos embaixadores são equivalentes.

```
//////////INICIO COMUNICAÇÃO COM O GATEWAY////////////////////////////////
Gateway.UpdateAttribute ("1.1",2, "0");
// Em federados 'notime' (FNT), o timestamp da msg é sempre "0"
//////////FIM COMUNICAÇÃO COM O GATEWAY////////////////////////////////
```

Figura 5.14: Exemplo de interface entre gateway e federado notime

Na Figura 5.14 é apresentado um exemplo de interação (uso de atributo de saída) entre federado *notime* ativo e respectivo *gatewayN*. Da mesma forma que em federados síncronos, o mesmo procedimento é adotado para qualquer federado destino. Contudo, o

federado executa apenas um tipo de interação com o *gatewayN*, pois não tem preocupação com o tempo, ao contrário do exemplo apresentado na Figura 5.13.

5.4 Integração de federados descritos em C/C++

Federados descritos na linguagem C/C++ podem cooperar diretamente com o *gatewayN* respectivo. Para isso, a comunicação é realizada através de rotinas que executam chamadas JNI (*Java Native Interface*). A Figura 5.15 apresenta um exemplo de código do federado 'Driver do Display' utilizado nos experimentos com o DCB (Capítulo 6). Nesta figura, as linhas 07 e 10 exemplificam a declaração de atributos de entrada e saída, respectivamente, na interface com o *gateway* do DCB.

Para a passagem de dados do *gateway* para o federado, o *gateway* instancia o método *Java_Gateway6_SendSignal* (linha 13 da Figura 5.15) declarado como interface JNI.

```
01 #include <stdio.h>
02 #include <conio.h>
03 #include <stdlib.h>
04 #include "jni.h"
05 #include "Gateway6.h"
06 // ENTRADAS
07 jint reset;
08 ...
09 // SAÍDAS
10 jint lcd_data;
11 ...
12 // MÉTODO NATIVO
13 JNIEXPORT void JNICALL Java_Gateway6_SendSignal
       (JNIEnv *env, jclass cls, jint reg_sel, jint inputdt)
14 {
15     jmethodID mid = env->GetStaticMethodID
              (cls, "UpdateAttribute","(Ljava/lang/String;II)V");
16     ...
17     jmethodID updLVT = env->GetStaticMethodID(cls, "updateLVT","(I)I");
18     ...
19     jmethodID retGVT = env->GetStaticMethodID(cls, "returnGVT","()I");
20      ...
21     if (initializing_flag)
22     {
23             tst=(env->CallStaticIntMethod(cls, retGVT));
24     ...
25             env->CallStaticVoidMethod
               (cls, mid, env->NewStringUTF("1.0"),inputdata, tst);
26     ...
27             drvlvtaux = env->CallStaticIntMethod(cls, updLVT, drvlvt);
28     ...
29     return;
30 }
```

Figura 5.15: Exemplo de código de um federado em C++

Para a passagem de dados do federado para o *gateway*, o federado inicialmente declara um rótulo para cada método do *gateway* que será instanciado:

- Linha 15 da figura declara *mid* para instanciação do método *UpdateAttribute* do *gateway;*
- Linha 17 da figura declara *updLVT* para instanciação do método *updateLVT* do *gateway;*
- Linha 19 da figura declara *retGVT* para instanciação do método *returnGVT* do *gateway.*

As linhas 23, 25 e 27 da figura apresentam um exemplo de instanciação dos métodos (do *gateway*) *returnGVT, UpdateAttribute* e *updateLVT*, respectivamente.

5.5 Integração de federados descritos em SystemC

Componentes IP descritos em SystemC podem ser integrados em modelos de co-simulação através do uso de três entidades de apoio: um módulo complementar do federado SystemC; o *gateway* (de acordo com a estrutura do DCB); e um driver de simulação.

Com o objetivo de integrar componentes SystemC, cada uma de suas portas é conectada, através de um sinal, com uma correspondente porta do módulo complementar. No exemplo ilustrado na Figura 5.16 as portas ‘a‘, ‘b‘, ‘c‘, e ‘d‘ do componente são conectadas com as portas correspondentes do módulo complementar através dos sinais A, B, C, e D. O módulo complementar pode ser gerado automaticamente pela ferramenta de configuração do Tangram através de um *template*.

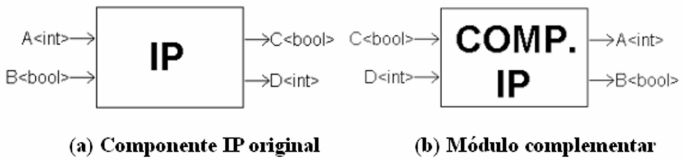

Figura 5.16: Componente IP em SystemC e módulo complementar

O *gatewayN* de um federado SystemC é similar ao *gatewayN* utilizado em federados descritos em C++. A principal diferença é a existência de um método auxiliar denominado ‘*getAttributeN*()’ para cada atributo de entrada do federado. Com estes métodos, um federado SystemC pode monitorar a existência de novas informações nos atributos de entrada. Além disso, pela chamada de uma função ‘*SendSignal’*, o *gatewayN* inicializa o federado na simulação e passa o controle (da interface) ao federado para que ele execute a verificação dos atributos de entrada em busca de novos valores.

Ou seja, não é o *gatewayN* que executa chamadas aos métodos nativos do federado, mas sim o federado SystemC que monitora a chegada de valores nos atributos de entrada através de métodos disponíveis no *gatewayN*. A Figura 5.17 demonstra a principal parte do código do *gatewayN* para federados SystemC.

```
// lendo a dll
static {
        System.loadLibrary("ip");
}

// passando controle para a dll (federado) através da chamada de uma função
// native do federado
public static void SetPointer(ApplicationDCB pointer)
        {
                g.SendSignal();
        }

// declarando os métodos natives
    public native void SendSignal();

// método utilizado para verificar alterações nos sinais de entrada
public static boolean updateOccur()
{
        return updated;
}

// Métodos que recuperam valores dos atributos de entrada
// Existem X métodos getAtrN para um federado com X portas
// E.g. se o federado instancia o método getAttribute1(), ele recupera
// o valor armazenado pelo gateway em value_attribute1
public static int getAttribute1()
{
        return value_attribute1;
}
public static int getAttribute2()
{
        return value_attribute2;
}
public static int getAttributeN()
{
        return value_attributeN;
}
```

Figura 5.17: GatewayN para federados descritos em SystemC

O driver de simulação é responsável por conectar as portas do componente descrito em SystemC com as portas do respectivo módulo complementar, atualizando seus atributos de saída e buscando os novos valores dos atributos de entrada através de métodos do *gatewayN* especificados para as portas do federado respectivo. Para este driver, uma biblioteca DLL precisa ser gerada e integrada à federação. O driver de simulação declara uma função nativa usando JNI de modo similar a federados descritos em C++. A Figura 5.18 ilustra a parte principal do código do driver de simulação para o componente IP 'driver do display' do experimento 4 (apresentado no Capítulo 6).

```
#include “systemc.h”
#include “ip.h”
#include “ipcomplementar.h”

JNIEXPORT void JNICALL Java_Gateway_initSimulation(JNIEnv *env, jobject obj) {

// declarando sinais
sc_signal<int> A, D;
sc_signal<bool> B, C;

// conectando as portas
ip i("ip");
i.a(A); i.b(B); i.c(C); i.d(D);

complementaryIP ipc(“complementaryIP”);
ipc.a(A); ipc.b(B); ipc.c(C); ipc.d(D);

// inicializando simulação
sc_initialize();

// identificando métodos do gateway
jmethodID midUpOccur = env->GetMethodID("updateOccur");
jmethodID midGetAt1 = env->GetMethodID("getAttribute1");

// laço de simulação para manter verificação por novas entradas
        while(true){
// verifica a existência de alterações nas entradas
        updated = env->CallStaticBooleanMethod(midUpOccur);
        if (updated != 0)
        {
                // se existe alteração, recupera os valores novos das portas de
entrada
                port_value = env->CallStaticIntMethod(midGetAt1);
                // e os atribui aos respectivos sinais
                input_data = port_value;
                // repete os dois commandos acima para todas as portas de entrada

                // em seguida, atualiza todos os sinais de entrada,
                // executando um ciclo de simulação
                sc_cycle(1);
                // e atualiza os sinais de saída do federado (se existirem)
                env->CallStaticVoidMethod(output_signal);
        }
}
}
```

Figura 5.18: Driver de simulação para federados descritos em SystemC

5.6 Integração de federados descritos em VHDL

Componentes descritos em VHDL podem ser integrados em modelos de co-simulação desde que seja estabelecido um meio de comunicação entre o *gateway* (de acordo com a estrutura do DCB) e uma ferramenta de suporte à execução de descrições VHDL. Esta seção discute a interligação com a ferramenta *ModelSim*.

A ferramenta *ModelSim* pode ser interligada ao DCB através de módulos de apoio, descritos em C e Java, como instrumento de ligação entre o *gateway* e uma interface de comunicação do *ModelSim* denominada FLI (*Foreign Language*

Interface). FLI é uma interface de comunicação que permite descrever o comportamento de um módulo em VHDL utilizando código objeto gerado a partir de um programa escrito em C.

No trabalho de integração de um componente em VHDL com o uso de *ModelSim,* inicialmente é necessária a construção de um módulo de interface em VHDL para de definir as portas de entrada e saída do componente. Cada uma de suas portas é conectada, através de um sinal, com uma porta correspondente do módulo de interface. Este módulo possui propósitos similares aos do 'módulo complementar' usado na integração de componentes em SystemC.

O módulo de interface possui uma descrição em C que, através da FLI, reconhece as portas cuja conexão já existe entre o componente e o módulo de interface. No lado do DCB, esta descrição em C coopera com um módulo intermediário (em Java) com quem o *gateway* realiza a troca de dados. A comunicação entre a descrição em C do módulo de interface e o módulo intermediário pode ser mantida com o uso de *sockets* ou JNI, por exemplo.

```
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------------------------------------
-- Esta descrição define a interface do componente descrito em
-- VHDL (portas de comunicação que são identificadas
-- pelo código em C através da FLI)
-------------------------------------------------------------------
entity interface is
  port(
        clk : in std_logic;
        reset : in std_logic;
        inputdata: out std_logic_vector(7 downto 0);
        lcd_data: in std_logic_vector(7 downto 0);
        done: in std_logic
      );
end interface;
architecture interface of interface is
       attribute foreign: string;
       attribute foreign of interface: architecture is "lcd_driver_init
lcd_driver.so";
begin
end interface;
```

Figura 5.19: Descrição intervace.vhd para federados descritos em VHDL

A Figura 5.19 apresenta o conteúdo do arquivo *interface.vhd* construído para um experimento que integra um componente em VHDL na federação utilizada como estudo de caso no escopo desta tese. Este experimento é equivalente ao experimento 4, apresentado no Capítulo 6, exceto pelo fato de que o federado 'Driver do Display' é descrito em VHDL e não em SystemC.

O módulo de interface é formado por *interface.vhd* em conjunto com o código C que utiliza FLI para reconhecer as portas de comunicação. Um trecho do código C utilizado no experimento 4 é apresentado na Figura 5.20. Nele é declarada uma estrutura de dados (*typedef struct*) equivalente à entidade declarada em *interface.vhd*, além de algumas funções para tratamento de dados e de comunicação com o módulo intermediário.

```
// estrutura que define a interface com a entidade em interface.vhd
typedef struct{
        signalID reset;
        driverID inputdata;
        signalID lcd_data;
        signalID done;
        SOCKET sock;
}signals_reg;

int g_input_array[12];
char g_buffer[10]={0,0,0,0,0,0,0,0,0,0};
//variável que armazena temporariamente os valores de saída
char g_input_data[8], g_lcd_data[8], g_reset, g_done;

void proc_clock_CB(signals_reg *sr);
void lcd_driver_init(regionID region, char *param, interface_list
      *generics, interface_list *ports);
// funções de comunicação com o módulo intermediário
void sockCB(signals_reg *sr);
void loadDoneCB(signals_reg *sr);
```

Figura 5.20: Código C que é parte do módulo de interface

A Figura 5.21 apresenta o código da função *sockCB* cujo propósito é de, a cada evento (comando *read* na figura), escrever o dado recebido pelo simulador *ModelSim* no sinal *sr->inputdata* através de *mti_ScheduleDriver*. Neste exemplo as linhas *sprintf*(..) e *mti_Command*(...) mantêm um intervalo de 40ns a cada interação na interface. A cada pacote recebido do DCB esta função é atualizada e executada novamente. A função *write*(...) tem o objetivo de transmitir valores de saída para o DCB via módulo intermediário.

```
void sockCB(signals_reg * sr)
{
        int i, value;
        char buf [10];
        char command_vsim[100];
        char str_value[10] = {' ',' ',' ',' ',' ',' ',' ',' ',' ',' '};
        // atribui a 'i' um dado recebido do módulo intermediário
        i = read( sr->sock, buf, sizeof(buf));
        convert_int_to_enums(atoi(buf), g_input_data, 8);
        // escreve dado no simulador ModelSim
        mti_ScheduleDriver( sr->inputdata, (long)g_input_data, 0, MTI_INTERNAL);
        sprintf( command_vsim, "run 40 ns");
        mti_Command(command_vsim);
        // recebe dados de saída...
        mti_GetArraySignalValue(sr->lcd_data, g_lcd_data);
        value = convert_enums_to_int(g_lcd_data, 8);
        sprintf(str_value,"%d\n",value);
        // ...e envia para o DCB
        if ((i = write(sr->sock, str_value , sizeof(str_value))) >= 0)     {
                printf("result sent to server\n");   }
        else    { printf("written error \n"); }
}
```

Figura 5.21: Detalhes da função sockCB

Por fim, a Figura 5.22 apresenta parte do código do módulo intermediário que mantém a interface com o *gateway* e se comunica com o módulo de interface.

```
private static ServerSocket s;
private static Socket incoming;
private static BufferedReader in;
private static PrintWriter out;

public Driver_Lcd() // Inicializa o consumidor
{
        int porta=25539;
        try {
                s = new ServerSocket(porta);
                System.out.println(" Servidor ativado na porta: " + porta);
                // aguarda pela chegada de dados
                incoming = s.accept();
                in = new BufferedReader(
                new InputStreamReader(incoming.getInputStream()));
                out = new PrintWriter(incoming.getOutputStream(),true);
        } catch(Exception e) { System.out.println("Erro =>" + e); }
}
public static void SendSignal(int reg_sel, int inputdt) {
...
        // repassa dados recebidos para o federado
        out.println(inputdt);
        try{ while (!feito){
                // recebe dados do federado
                input_data = in.readLine();
                ...
             }
             result_data = Integer.parseInt(input_data.trim());
             // envia dados para o DCB instanciando um método do gateway
             Gateway.UpdateAttribute("1.0",result_data);
        } catch (Exception e) {System.exit(0);}
        System.out.println("Enviando lcd_select => " + lcd_select);
        Gateway.UpdateAttribute("2.0",lcd_select);
        }
}
```

Figura 5.22: Exemplo de código do módulo intermediário

Neste exemplo, a comunicação entre o módulo de interface do *ModelSim* e o módulo intermediário é mantida com o uso de *sockets.* A linha *incoming = s.accept()* monitora a conexão. As linhas *input_data=in.readLine()* e *Gateway.UpdateAttribute*(...) recebem dados do federado e enviam ao DCB, respectivamente. A linha *out.println(inputdt)* repassa os dados de entrada para o federado.

O experimento que incorpora em uma federação um federado descrito em VHDL foi desenvolvido no contexto de uma bolsa DTI (CAPES).

6 EXPERIMENTOS COM PROTÓTIPO DO TANGRAM/DCB

6.1 Introdução

Este capítulo apresenta o desenvolvimento de experimentos com o uso do protótipo do Tangram/DCB apresentado neste trabalho. No desenvolvimento dos experimentos foram construídas e executadas quatro versões de um modelo de representação do SistemaGPSAlerta [LIS2002].

Os experimentos têm como objetivo geral a observação prática dos propósitos do DCB na co-simulação em termos de: preservação de detalhes internos dos federados; sincronização híbrida; encapsulamento das políticas de gerenciamento de tempo; e a distribuição física dos federados. A realização de parte dos experimentos foi feita no escopo do trabalho de dissertação de mestrado [SPE2003]. Atualmente, novos experimentos estão em andamento em conjunto com a extensão do protótipo.

Para mensurar o desempenho na cooperação entre componentes em execuções centralizadas e distribuídas, são apresentados resultados obtidos na simulação de quatro versões do modelo. Basicamente, tais resultados traçam um perfil do comportamento do DCB em relação ao tempo real gasto para executar uma mesma tarefa sob diferentes condições de distribuição e de composição/heterogeneidade do modelo.

Neste capítulo, os termos ‘simplicidade’ e ‘complexidade’ são utilizados com o propósito de indicar baixo ou alto grau de detalhamento do modelo (abstração) em relação ao sistema real que representa, conforme abordagem do Capítulo 2.

6.2 Apresentação do SistemaGPSAlerta

Instalado em meios de locomoção (veículos) e conectado a um dispositivo de GPS (*Global Position System*), o SistemaGPSAlerta [LIS2002] tem como objetivo monitorar a posição geográfica do objeto em que está instalado. Além do trabalho de monitoração, pontos geográficos podem ser previamente cadastrados no SistemaGPSAlerta que, mediante a proximidade do meio de locomoção destes pontos, emite alarmes. Para o disparo de alarmes o SistemaGPSAlerta executa comparações entre a posição atual e as posições cadastradas.

O terminal do SistemaGPSAlerta foi projetado para ser conectado a um Garmin GPS 31/31 SL TrackPak por meio de uma interface RS-232. A comunicação entre o GPS e o terminal ocorre através de mensagens ASCII.

6.3 Experimentos com modelos do SistemaGPSAlerta

Os experimentos com modelos do SistemaGPSAlerta têm como propósito essencial a demonstração das propriedades do protótipo do Tangram/DCB na validação das relações funcionais entre os federados que compõem um modelo. Neles, diferentes combinações entre os federados do modelo são construídas. Nestas combinações, os níveis de abstração dos federados são alterados de modo não uniforme. Ou seja, enquanto alguns federados têm sua interface refinada/alterada, outros são mantidos inalterados.

O processo de projeto e execução dos modelos reflete as funcionalidades gerais de modelagem descritas em conjunto com a apresentação dos requisitos para o Tangram (Capítulo 4).

Embora os arquivos dos *gateways* sejam gerados automaticamente pela ferramenta de configuração, em alguns casos foram feitas alterações explícitas nestes arquivos com o propósito de incluir características não previstas nos moldes (*templates*) existentes no protótipo atual. Tais alterações significam, por exemplo, a inclusão de uma linha de código para tradução de um tipo não previsto (p.ex. hexadecimal). O formato, as propriedades e características e a funcionalidade do *gateway* são mantidas.

Embora, para alterações explícitas, o projetista precise conhecer a estrutura do *gateway* para executar alterações que porventura sejam necessárias, as intervenções tendem a ser pequenas, em geral uma linha de código por primitiva de envio ou recebimento de valor. Não são necessárias alterações em nenhum outro módulo do DCB. Esta característica está relacionada à busca pela independência dos federados (flexibilidade/generalidade do DCB), reduzindo o esforço para sua inclusão numa federação.

Nos três primeiros experimentos o modelo do SistemaGPSAlerta foi executado sobre a versão não temporizada do DCB. O quarto experimento foi executado sobre a versão temporizada do DCB, versão que utiliza o federado 'Fedgvt' para gerenciamento do GVT.

Neste capítulo o termo '*gateway*' é utilizado para designar os arquivos que implementam o *gateway* e *gatewayN,* como descrito na Seção 5.3.1 desta tese.

6.3.1 Experimento 1

Neste experimento os detalhes internos dos módulos e questões de comunicação são descritos em um alto nível de abstração (Figura 6.1). Para isso, o modelo do SistemaGPSAlerta é composto por quatro componentes que interagem através de primitivas de comunicação de alto nível (p.ex. *send*, *receive*).

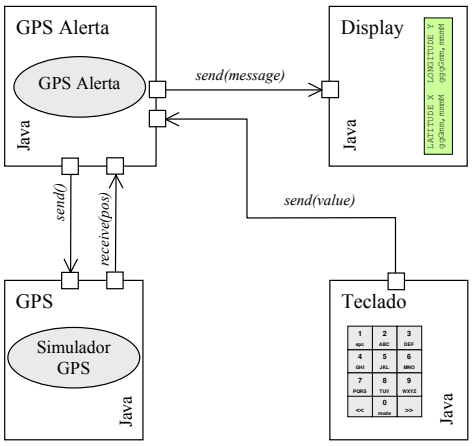

Figura 6.1: SistemaGPS Alerta modelado em um alto-nível de abstração

Implementados em Java, os federados deste modelo são: GPS Alerta; Simulador GPS; Teclado; e Display. Eles representam quatro diferentes tarefas, sendo [SPE2003]:

1. GPS Alerta – é o módulo principal do sistema. Tem como principais tarefas solicitar e receber dados do GPS, calcular e comparar posições geográficas, atender a chamadas do teclado e enviar dados para exibição no Display;
2. Simulador GPS – simula a geração de uma seqüência de coordenadas, da mesma forma como um GPS real. Essa aplicação, modelada em Java, lê coordenadas de um arquivo texto e as disponibiliza para o GPS Alerta, de acordo com a formatação descrita em [LIS2002];
3. Teclado – aplicação Java que simula um teclado numérico. O pressionamento de uma tecla implica na geração de um valor de saída, que é enviado para o GPS Alerta;
4. Display – aplicação Java que simula um Display gráfico de 2 linhas com 24 caracteres.

Mesmo tendo um conjunto reduzido de detalhes (simplicidade), este experimento já oferece suporte para validação das relações de cooperação entre os federados utilizados (funcionalidade). Além disso, facilita o entendimento do contexto geral do sistema. Esta característica tem sua importância aumentada proporcionalmente ao tamanho e complexidade do sistema que esteja sendo representado.

Tabela 6.1: Informações estáticas do experimento 1

Federado	Linguagem do federado	Tamanho do federado (linhas)	Tamanho do *gateway* gerado (linhas)	Template utilizado	Tamanho do XML gerado para config. do EF e EDCB (linhas)
GPS Alerta	Java	348	158	Java	17
Simulador GPS	Java	107	127	Java	12
Display	Java	439	145	Java	8
Teclado	Java	230	130	Java	12

A Tabela 6.1 apresenta informações estáticas sobre a federação utilizada neste primeiro experimento. Uma observação relevante, válida também para os próximos experimentos, é que o tamanho do arquivo do *gateway* de cada federado não depende do tamanho do federado. Observa-se que os tamanhos não variam proporcionalmente. O tamanho do *gateway* varia apenas em função do número de atributos de entrada e de saída do federado que representa. Mesmo assim, o impacto no arquivo é bastante reduzido, como discutido na Seção 5.3.1. O mesmo é válido para o arquivo de configuração (XML).

6.3.2 Experimento 2

Neste segundo experimento, o GPSAlerta deixa de ser modelado como um componente do sistema, e passa a ser visto como uma tarefa executada por um micro-controlador, este visto como componente. O micro-controlador utilizado é o FemtoJava [ITO2001], cuja arquitetura é representada na Figura 6.2.

O federado que representa o comportamento do FemtoJava é descrito em Java. O modelo de comunicação disponibilizado por este federado permite que a interação com os componentes (com os quais coopera) ocorra através de dois principais meios: portas de entrada e saída (E/S) ou interrupções. As portas de E/S são utilizadas respectivamente para leitura e escrita de valores, já as chamadas de interrupção são utilizadas para interromper a execução do micro-controlador e, posteriormente, executar um determinado procedimento.

Com a inclusão deste federado, a comunicação passa a ser mantida por um protocolo de mais baixo nível que utiliza tipos de dados específicos (e.g. *float, int, string*). Neste modelo, apenas alguns dos recursos do micro-controlador são utilizados. No entanto, a inclusão de detalhes de funcionamento e de comunicação do SistemaGPSAlerta permite que sejam validadas relações funcionais inexistentes no modelo anterior.

Tomando como exemplo o federado Simulador GPS, observa-se que a mudança do nível de abstração do federado GPS Alerta, devido à inclusão do FemtoJava, não interferiu no grau de detalhamento e na interface do Simulador GPS. Este é um exemplo de convivência entre dois níveis de abstração num mesmo modelo.

O DCB provê esta característica de modo inerente, pois não requer que um federado reconheça as características/propriedades de outros federados, nem mesmo a sua existência. Assim, neste exemplo, o DCB continua reconhecendo a interface do Simulador GPS da mesma forma que no exemplo anterior (protocolo *send/receive*), enquanto trata a interface do FemtoJava como portas de entrada e saída (portas IN3 e OUT3), como mostrado na Figura 6.3.

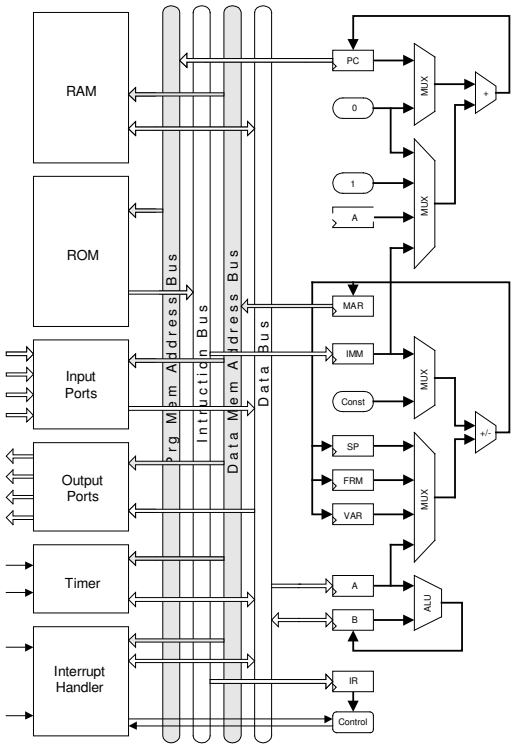

Figura 6.2: Detalhes do micro-controlador FemtoJava

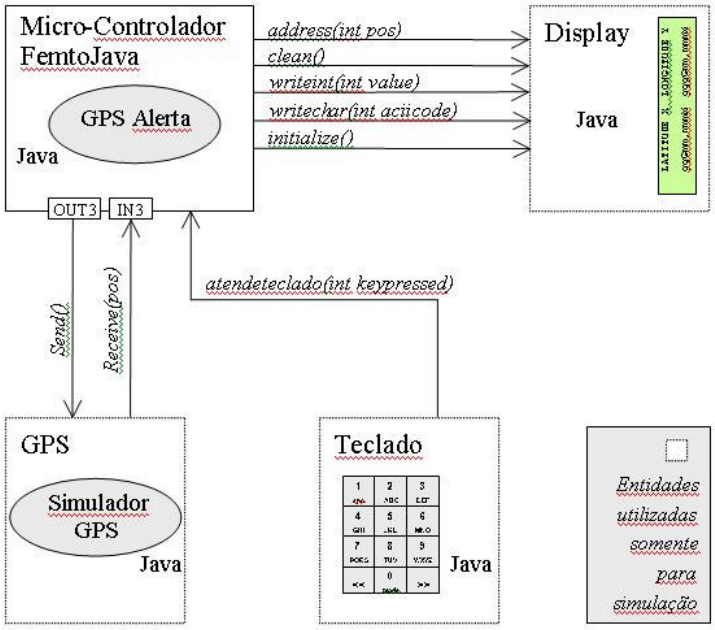

Figura 6.3: Modelo do SistemaGPSAlerta utilizado no experimento 2

Em comparação com o modelo comportamental (descrito no experimento anterior), os demais componentes do sistema (Display e Teclado) não possuem alterações significativas na modelagem de seu comportamento. As principais alterações

estão localizadas em suas interfaces, que sofreram um primeiro processo de refinamento no protocolo de comunicação, conforme detalhado abaixo:

- Display – o conjunto de funções da interface desse componente foi mais bem detalhado. Foram adicionadas funcionalidades responsáveis pela inicialização do Display, limpeza da tela, posicionamento do cursor e escrita de dados;
- Teclado – sofreu um pequeno refinamento na comunicação, passando a utilizar um tipo de dado específico (int) no evento de notificação do GPS Alerta.

Tabela 6.2: Informações estáticas do experimento 2

Federado	Linguagem do federado	Tamanho do federado (linhas)	Tamanho do *gateway* gerado (linhas)	Template utilizado	Tamanho do XML gerado para config. do EF e EDCB (linhas)
GPS Alerta	Java	350	158	Java	27
Simulador GPS	Java	107	127	Java	12
Display	Java	600	203	Java	11
Teclado	Java	230	130	Java	12

A Tabela 6.2 apresenta informações estáticas sobre a federação utilizada neste segundo experimento. As considerações feitas para a Tabela 6.1 são válidas também para esta tabela. Entre elas, observa-se que o tamanho do *gateway* e do XML é proporcional às alterações no número de atributos de interface dos federados, e não do seu tamanho.

6.3.3 Experimento 3

Neste experimento o tratamento de um número maior de detalhes já permite que o modelo represente com maior fidelidade (detalhamento) a forma da arquitetura final do sistema real, embora ainda mantenha um certo nível de abstração se comparado ao nível de implementação. A Figura 6.4 ilustra os componentes, e suas interligações, deste experimento.

Em comparação com o modelo do experimento 2, este modelo sofreu a adição de novos componentes descritos em linguagem diferente, além de novas alterações nas interfaces de comunicação. As alterações são descritas nos itens a seguir [SPE2003]:

- Driver de hardware do Display – foi adicionado ao modelo para executar as tarefas relacionadas ao gerenciamento do hardware do Display real. É baseado em uma FSM (Finite State Machine) originalmente descrita em VHDL. Esse modelo VHDL [SAS2004] foi obtido junto aos códigos que compõem o FemtoJava original e foi reescrito em C++. A comunicação com o gateway é feita através de uma interface JNI (Java Native Interface);

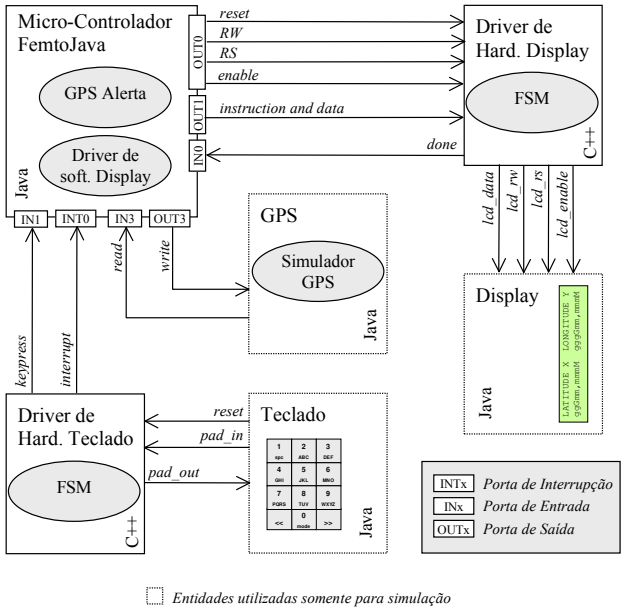

Figura 6.4: Modelo do SistemaGPSAlerta utilizado no experimento 3

- Driver de hardware do Teclado – foi inserido no modelo para controlar o hardware do Teclado real. Assim como o Driver do Display, esse componente é baseado em máquina de estados finitos. O Driver do Teclado é responsável pela varredura e identificação da tecla pressionada pelo usuário, notificação do GPS Alerta através de uma chamada à interrupção INT0 do FemtoJava e posterior escrita do valor da tecla pressionada na porta IN1. A máquina de estados implementada nesse modelo foi construída a partir de um modelo VHDL já existente e reescrita em C++ utilizando uma interface JNI para se comunicar com a infra-estrutura de co-simulação;
- Driver de software do Display – módulo introduzido no modelo (no federado FemtoJava) com o objetivo de disponibilizar ao GPS Alerta funções de alto nível para controle do Display. A cooperação desse módulo com o driver de hardware do Display é realizada através das portas de saída OUT0 e OUT1 do FemtoJava;
- Display – esse componente sofreu um refinamento em sua interface de comunicação. O conjunto de sinais de entrada foi alterado para prover uma simulação mais realista do modelo e para torná-lo compatível com o driver de hardware utilizado.
- Teclado – Assim como ocorreu com o display, a interface de comunicação do Teclado também sofreu um processo de refinamento, oferecendo recursos para o Driver do Teclado executar o processo de varredura, que resulta na identificação da posição (linha/coluna) da tecla pressionada;
- Simulador GPS – adaptado ao modo de comunicação do FemtoJava, agora executa leitura e escrita nas portas de entrada e saída. Recebe uma solicitação através da porta OUT3 e disponibiliza um novo dado para leitura na porta IN3.

Neste experimento, a descrição do componente GPS Alerta, ainda em Java, agora combina uma especificação computacional abstrata com uma comunicação

refinada (nível RT) com a inclusão do 'Driver de hardware do Display'. A especificação computacional é descrita pela funcionalidade do GPS Alerta. A comunicação, contudo, considera a interface real do microcontrolador, consistindo de portas de entrada e saída e de sinais de interrupção. A mesma modelagem, combinando níveis em Java, permanece para os componentes 'Simulador GPS', 'Teclado', e 'Display'. Embora ainda abstrata se comparado ao nível de implementação do sistema, a cooperação entre os componentes já reproduz de forma fiel o modo como se dá a comunicação entre os componentes reais.

Este experimento também incorpora a simulação heterogênea pelo fato de que suporta a cooperação entre componentes descritos em linguagens diferentes. Observa-se que toda modificação necessária para inclusão desses componentes permanece restrita aos aspectos de interface (promovendo independência de federados). Ainda, o suporte à co-simulação mantido pelo DCB (implementação do DCB) não sofre nenhuma alteração. Apenas a atividade de configuração é realizada a partir de dados descritos em XML. O mesmo procedimento é mantido em qualquer nível de abstração, em qualquer grau de heterogeneidade (número, forma e tecnologia de componentes, etc).

Na Tabela 6.3 observa-se aumento de tamanho do *gateway* no tratamento de interfaces heterogêneas. Contudo, são igualmente válidas as afirmações descritas na apresentação dos experimentos anteriores sobre o esforço necessário para construção do *gateway* e também da limitação da necessidade de alterações aos aspectos de interface apenas.

Tabela 6.3: Informações estáticas do experimento 3

Federado	Linguagem do federado	Tamanho do federado (linhas)	Tamanho do *gateway* gerado (linhas)	Template utilizado	Tamanho do XML gerado para config. do EF e EDCB (linhas)
GPS Alerta	Java	345	159	Java	17
Simulador GPS	Java	107	127	Java	12
Display	Java	440	146	Java	8
Driver do display	C++	202	243 *	JNI	17
Teclado	Java	288	146	Java	16
Driver do teclado	C++	364	254 *	JNI	18

* Inclui o código C++ que fornece acesso as funções da *Java Native Interface*.

6.3.4 Experimento 4

A Figura 6.5 apresenta a arquitetura do SistemaGPSAlerta com a inclusão de um federado descrito em SystemC [PAN2001]. Três componentes são reutilizados de versões anteriores da federação: o microcontrolador Java [ITO2001], o Teclado e os drivers para conectar o microcontrolador a periféricos. De acordo com as propriedades de co-simulação do DCB, a distribuição dos federados pode ser realizada sem contudo comprometer a funcionalidade da simulação. No entanto, é natural que o desempenho da simulação seja reduzido proporcionalmente à distribuição do modelo em nodos distintos.

Por exemplo, se o federado Simulador GPS for alocado a um nodo remoto em relação ao restante da federação, o número de mensagens que farão uso de serviços de rede será pequeno. No entanto, se o FemtoJava for alocado a um nodo remoto, o

impacto no desempenho será expressivo. Em ambas as situações a funcionalidade não é afetada. Esta característica é válida para todos os experimentos (discutida na próxima seção).

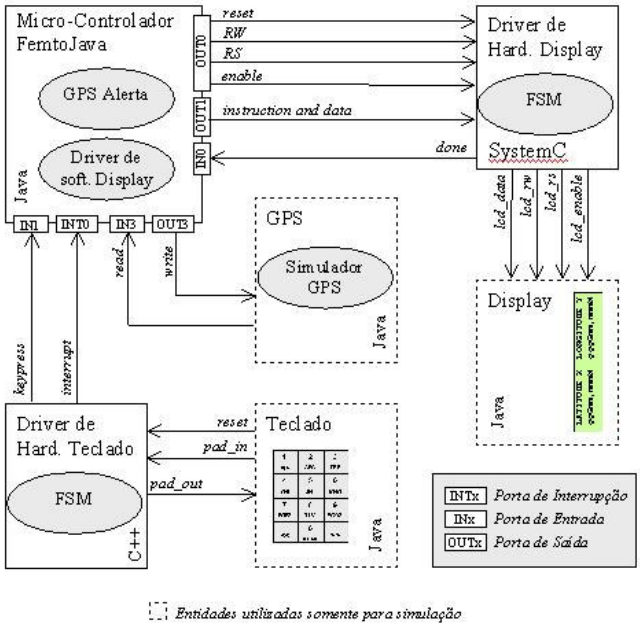

Figura 6.5: Modelo do SistemaGPSAlerta utilizado no experimento 4

Neste experimento o Driver de Teclado é descrito em C++ e o Driver de Display é descrito em SystemC e implementa uma interface proprietária. Com estes componentes tem-se uma federação heterogênea e distribuída (Java/C++/SystemC). A Tabela 6.4 apresenta o número de linhas dos federados e respectivos *gateway*s.

Observa-se, na Tabela 6.4, que mesmo para federados mais complexos, os *gateways* mantêm o tamanho reduzido. Ou seja, neste caso são igualmente válidas as afirmações descritas na apresentação dos experimentos anteriores sobre o esforço necessário para construção do *gateway*. Também é válida a afirmação de que o seu tamanho é proporcional ao número de sinais de interface do componente que representa, e não ao tamanho do componente. De modo equivalente aos experimentos anteriores, os embaixadores e o núcleo do DCB permaneceram inalterados.

Tabela 6.4: Informações estáticas do experimento 4

Federado	Linguagem do Federado	Tamanho do Federado (linhas)	Tamanho do *gateway* gerado (linhas)
Fedgvt	Java	81	140
GPS Alerta	Java	349	159
Simulador GPS	Java	107	120
Display	Java	398	138
Driver do display	SystemC	269	218 * *
Teclado	Java	208	152
Driver do Teclado	C++	309	243 *

* inclui o código C++ que dá acesso ao JNI

** inclui o driver de simulação (75 linhas) e o modulo complementar (11 linhas)

Como neste experimento o modelo do SistemaGPSAlerta foi executado com o uso da versão temporizada do DCB. Com isso passa a ser utilizado o federado Fedgvt que mantém o gerenciamento do GVT.

Este experimento também promove a sincronização híbrida. Nele, o Simulador GPS foi configurado para executar no modo síncrono e os demais federados no modo *notime.* Na execução dos testes de desempenho a versão similar à construída neste experimento teve o Display configurado como *notime* e os demais federados como síncronos sem alteração do comportamento do modelo.

6.3.5 Considerações sobre os experimentos

A realização dos experimentos permitiu observar o DCB em relação aos propósitos de independência entre federados, sincronização híbrida, políticas de gerenciamento e aspectos de generalidade e flexibilidade.

Na transição do experimento 3 para o experimento 4 foi realizada a inclusão de um novo federado denominado Driver do Display numa posição intermediária aos federados Micro-controlador FemtoJava e Display (já existentes). Embora a inclusão deste novo federado tenha provocado alterações na troca de dados, o comportamento do Display foi preservado na íntegra, tendo sofrido ajustes na interface apenas. No caso da impossibilidade de se fazer ajustes na interface do Display, o *gateway* poderia ser responsabilizado pela tarefa de traduzir os dados para o formato reconhecido no destino (no caso deste exemplo o Display apenas recebe dados). Um outro exemplo de inclusão de um federado pode ser observado na transição do experimento 2 para o experimento 3 onde o novo federado é denominado GPS. A inclusão de um federado sem comprometimento da federação é vinculada à característica de independência entre federados promovida pelo DCB.

A independência entre federados também permite a convivência entre diferentes níveis de abstração. Por exemplo, na transição do experimento 2 para o 3 o federado Micro-Contolador FemtoJava sofreu a inclusão de um novo módulo cuja cooperação com o federado Driver do Display passou a ser realizada através de portas ‘in’ e ‘out’. Também neste experimento observa-se que o Driver do Display está implementado em uma linguagem distinta dos federados com quem coopera. No entanto, nenhuma alteração motivada pela heterogeneidade foi feita nos federados com quem este novo federado coopera. Ou seja, a simulação heterogênea é inerente à estrutura do DCB.

Particularmente no experimento 4, onde foi utilizada a versão temporizada do DCB, foi possível observar a cooperação entre federados que utilizam modos distintos de sincronização. No caso deste experimento ocorre cooperação entre federados síncronos e *notime*. Como federados *notime* não geram mensagens de gerenciamento para atualização de GVT, seu uso tende a melhorar o desempenho geral de uma simulação. Faz parte do trabalho de modelagem (projetista) identificar o modo de sincronização mais adequado para cada federado. Em geral, é mais adequado que federados passivos (que não enviam mensagens) sejam configurados como *notime.*

Com o encapsulamento das atividades de gerenciamento da simulação, a mesma implementação do DCB é utilizada nos experimentos 1,2 e 3. Ou seja, a construção de um novo modelo ou a mudança de um modelo existente não interfere na implementação dos módulos do DCB (EDCB, EF e NDCB), mas apenas no *gateway*. Esta propriedade

atribui flexibilidade e generalidade ao DCB, e também contribui com a independência dos federados. A versão temporizada do DCB, utilizada no experimento 4, possui a mesma propriedade. Nos testes de desempenho, apresentados na próxima seção, duas das versões do modelo do SistemaGPSAlerta utilizam o DCB não temporizado e três o DCB temporizado.

O experimento 4 também inclui um federado descrito em uma terceira linguagem, SystemC. De modo equivalente à inclusão do federado em C, nenhuma extensão foi realizada no DCB ou no federado em SystemC para o suporte ao novo federado. As alterações ficaram restritas aos aspectos de interface do federado e do *gateway* (generalidade).

Foi realizado um quinto experimento que exemplifica a inclusão de um federado descrito em VHDL executando sobre *ModelSim*. Ele é similar ao experimento 4, exceto pelo fato de que o federado 'Driver do Display' é substituído por uma versão em VHDL. A substituição deste federado não exigiu alterações nos demais federados, apenas o *gateway* foi adaptado para a interface do novo federado. Deste modo, as mesmas afirmações sobre sincronização híbrida, tamanho do *gateway*, independência dos federados, entre outras feitas nos experimentos anteriores, são igualmente válidas. A estratégia utilizada para integração de federados VHDL com o uso de *ModelSim* é discutida na Seção 5.6 desta tese.

6.4 Testes de desempenho na execução de modelos do SistemaGPSAlerta

Foram realizadas medidas de tempo de execução dos modelos do SistemaGPSAlerta construídos durante os experimentos com o propósito de estimar o desempenho do DCB. Com o objetivo de obter informações passíveis de comparação entre a execução das diferentes versões da federação do SistemaGPSAlerta foram considerados os seguintes itens: período de observação, número de mensagens, configuração da federação, e plataforma de execução.

O período de observação tem seu início no evento de envio da primeira mensagem de um 'grupo' a partir do federado origem e seu fim na 'entrega' da última mensagem do grupo ao federado destino. Neste caso, um 'grupo' significa o conjunto de mensagens necessário para uma atualização da interface implementada pelo federado Display. O termo 'entrega' significa o repasse de uma mensagem ao federado destino pelo seu respectivo *gateway*.

Em todas as versões utilizadas nas medidas de desempenho o período de observação compreende o tempo gasto para o envio de um grupo de mensagens tendo o federado GPS Alerta como origem e o federado Display ou Driver do Display como destino. O primeiro (Display) é considerado em versões com todos os federados implementados em Java e o segundo em versões com o Driver do Display em C ou em *SystemC*.

Quanto à configuração das federações, duas versões foram excutadas sobre o protótipo do DCB que não possui tratamento de tempo implementado (não temporizado): uma delas com todos os federados em Java, e a outra com o Driver do Display em C e os demais federados em Java.

As outras três versões foram executadas sobre a implementação do DCB com tratamento de tempo (temporizado), uma delas com todos os federados em Java, a segunda com o Driver do Display em C com os demais em Java e a terceira com o Driver do Display em SystemC com os demais federados em Java. A Tabela 6.5 apresenta a configuração de cada uma das versões utilizadas.

Tabela 6.5: Configuração das federações

Versão	Federados em Java	Federados em C	Federados em SystemC	DCB temporizado
Java-NT	GPS GPS Alerta Teclado Display			Não
JavaC-NT	GPS GPS Alerta Display	Driver do Display		Não
Java-T	GPS GPS Alerta Teclado Display			Sim
JavaC-T	GPS GPS Alerta Teclado Display	Driver do Display		Sim
JavaSyC-T	GPS GPS Alerta Teclado Display		Driver do Display	Sim

No nome de cada versão da tabela o NT identifica que o modelo foi executado com o uso do DCB não temporizado e o T identifica que o modelo foi executado com o uso do DCB temporizado. A descrição anterior ao símbolo '-' identifica as linguagens utilizadas pelos componentes da versão da federação.

O número de mensagens varia entre as diferentes federações construídas para o SistemaGPSAlerta. Por exemplo, em federações síncronas onde o GPS Alerta envia mensagens ao Display, uma atualização requer 42 mensagens úteis (geradas pelo modelo) e 9 mensagens de gerenciamento de tempo (geradas pelo DCB). O número de mensagens de cada uma das versões é apresentado na Tabela 6.6.

A comunicação entre o GPS Alerta e o Display ou o Driver do Display, de acordo com a versão, utiliza *sockets* em todas as execuções para tomada de desempenho (local e distribuída, temporizada e não temporizada).

Nos testes, cada versão foi simulada com todos os federados executando num único nodo e em seguida em dois nodos interconectados e isolados dos demais nodos da rede local do laboratório utilizado nessa tarefa. No uso de dois nodos o GPS Alerta foi executado em nodo distinto do respectivo federado destino (Display ou Driver do Display, dependendo da versão).

Como recurso de hardware foram utilizadas duas máquinas com processador Pentium 4 1.6Ghz, 128 RAM e conectadas com adaptador de rede a 100 Mbps.

6.4.1 Resultados das medidas de desempenho

A Tabela 6.6 apresenta os resultados obtidos com a execução de cada uma das versões do modelo do SistemaGPSAlerta. Para cada uma delas a coluna 2 da tabela apresenta o número de mensagens úteis (geradas pelos federados do modelo), a coluna 3 apresenta o número de mensagens de controle (geradas pelo DCB) e as colunas 4 e 5 apresentam o tempo gasto (em milisegundos) para o envio de um 'grupo' de mensagens na execução local e distribuída, respectivamente. A coluna 1 apresenta os federados origem e destino em cada versão.

O tempo (em *ms*) apresentado nas colunas 4 e 5 da tabela é resultante da média aritmética de uma coleta com 50 observações, desconsiderados os 5 maiores e os 5 menores tempos obtidos. Uma observação corresponde ao envio de um grupo de mensagens entre federado origem e destino, de acordo com a versão. Alguns dos tempos coletados destoam muito do comportamento médio (essencialmente no início da execução), motivo pelo qual alguns dos tempos foram desconsiderados.

Tabela 6.6: Consumo de tempo (em ms) na comunicação entre GPS Alerta e Display/Driver do Display

	1	*2*	*3*	*4*	*5*
Versão	**Comunicação: de – para**	**Número de msgs úteis**	**Número de msgs de controle**	**Simulação local (ms)**	**Simulação distribuída (ms)**
Java-NT	GPS Alerta – Display	42	42	46	10.000
JavaC-NT	GPS Alerta – Driver do Display	282	282	412	82.000
Java-T	GPS Alerta – Display	42	9	60	320
JavaC-T	GPS Alerta – Driver do Display	282	9	40	442
JavaSyC-T	GPS Alerta – Driver do Display	282	9	79	304

Na Tabela 6.6 observa-se a diferença no número de mensagens de controle nas versões dos modelos que utilizam o DCB não temporizado (Java-NT e JavaC-NT) em comparação com as versões que utilizam o DCB temporizado (Java-T, JavaC-T e JavaSyC-T). Nas versões com o DCB não temporizado cada mensagem útil enviada exige o retorno de uma mensagem de confirmação de recebimento do federado destino. Além disso, enquanto a confirmação não é recebida pela origem a próxima mensagem não é enviada. Deste modo, além do aumento do número de mensagens, a espera pela chegada da confirmação para o envio da próxima mensagem aumenta bastante o tempo de envio de um 'grupo' de mensagens.

Este fato pode ser observado tanto nas execuções locais quanto nas distribuídas, exceto na execução local da versão Java-NT que é mais rápida que a execução local da versão Java-T (esta sobre o DCB temporizado). Neste caso, para um número pequeno de mensagens, a execução local pode ser mais rápida com o uso do DCB não temporizado. No entanto, com o uso do DCB não temporizado, o aumento do número de mensagens provoca uma queda proporcional no desempenho (execuções locais das versões Java-NT e JavaC-NT), o que não acontece com o uso do DCB temporizado (execuções locais das versões Java-T e JavaC-T). No caso da versão JavaC-T ocorre, inclusive, uma redução do tempo necessário em função de que o federado destino está descrito na linguagem C.

Num comparativo entre as execuções local e distribuída das versões Java-NT e Java-T, o tempo necessário aumenta em torno de 200 vezes na primeira e 5 vezes na

segunda. Durante os testes de desempenho o percentual de consumo de processamento das máquinas se manteve baixo. Isto confirma o fato de que o consumo de tempo está no esforço de comunicação, e não no processamento dos federados, e que a versão não temporizada do DCB se apresenta inviável em comparação com a versão temporizada num ambiente distribuído. Mais do que isso, considerando a diferença no número de mensagens, na execução distribuída das versões Java-NT e JavaC-NT o consumo de tempo aumentou na mesma proporção (em torno de 200 vezes). Ou seja, em ambos os casos são enviadas em torno de 3,6 mensagens por segundo.

Nas versões que utilizam o DCB temporizado, a atividade de envio de mensagens atribui a cada uma delas um selo de tempo (*timestamp*) que determina a ordem de 'entrega' da mensagem ao federado destino. Tanto a atribuição do selo de tempo na origem quanto a entrega da mensagem no destino têm como referência o tempo global mantido pelo NDCB em conjunto com o federado 'Fedgvt'.

Esta alternativa reduz o número de mensagens de controle e não impõe restrições à atividade de envio de mensagens nos nodos origem. Ou seja, não é necessário nenhum tipo de espera para que uma próxima mensagem possa ser enviada. A diferença de desempenho entre ambas as formas de gerenciamento (DCB temporizado e não temporizado) da simulação é mais expressiva na comunicação distribuída (coluna 5 da Tabela 6.6).

Entre as execuções das versões JavaC-T e JavaSyC-T, particularmente, observa-se uma inversão da proporcionalidade de tempo na comparação entre as execuções locais e distribuídas. Ou seja, embora a versão JavaC-T tenha apresentado um desempenho melhor na execução local do que a versão JavaSyC-T, a versão JavaSyC-T apresentou melhor desempenho do que a versão JavaC-T na execução distribuída (Tabela 6.6). Este fato se deve ao modo como foram implementadas as duas versões do federado Driver do Display nas versões JavaC-T (implementado em C) e JavaSyC-T (implementado em SystemC).

Na versão JavaC-T é o *gateway* do Driver do Display que instancia um método implementado no federado para efetivar a entrega de uma mensagem. Deste modo, não há processamento no federado enquanto não houver mensagens a serem entregues. Já na versão JavaSyC-T é o federado que busca constantemente (laço) no respectivo *gateway* a existência de novos valores de entrada. Esta busca repetitiva executada pelo federado aumenta bastante o consumo de processamento do nodo, reduzindo o desempenho dos demais processos em execução neste mesmo nodo. Apesar disso, a execução da federação não entra em situação de impasse em função do serviço de escalonamento de processos mantido pelo sistema operacional [TAN2003].

Assim, na execução local da versão JavaSyC-T, o consumo de processamento do Driver do Display interfere no desempenho do federado GPS Alerta que é reponsável pelo envio dos grupos de mensagens. Este fato torna a execução local mais demorada em comparação com a execução local da versão JavaC-T.

No entanto, na execução distribuída da versão JavaSyC-T, o Driver do Display é executado em um nodo distinto, o que reduz a carga de processamento do nodo onde são executados os demais federados. Da mesma forma, o Driver do Display tem à disposição mais recursos de processamento para a busca de dados de entrada junto ao seu *gateway*. Este fato torna a execução distribuída mais rápida em comparação com a execução distribuída da versão JavaC-T.

Além das medidas realizadas com o uso de *sockets* na comunicação entre federados (local e distribuída), foi realizada uma medida de desempenho com a versão JavaC-T em uma execução local sem o uso de *sockets* e sob as mesmas condições. Nesta execução a comunicação ocorre através da chamada de métodos, disponíveis no DCB, para a troca de mensagens entre federados localizados no mesmo nodo e que executam em *threads* distintas em uma mesma JVM. Esta execução local tem como objetivo observar o comportamento do DCB considerando o consumo de tempo para a chamada de métodos de comunicação em substituição ao uso de *sockets*.

Como resultado, em média, o envio de um grupo de mensagens consome 32 ms, 8 ms a menos que na execução local com o uso de *sockets*. Como o mecanismo de comunicação do DCB é o mesmo nas demais versões temporizadas, podendo variar apenas a quantidade de processamento que o federado GPSAlerta utiliza para gerar cada mensagem, a tendência é de que a redução do consumo de tempo se mantenha também em torno de 25%.

No desenvolvimento do protótipo do DCB, até o momento, os esforços foram direcionados principalmente para a simulação heterogênea e híbrida (em relação ao gerenciamento de tempo). No entanto, pesquisas em simulação distribuída têm desenvolvido alternativas que contribuem com a melhoria de desempenho. Por exemplo, na redução do número de mensagens de simulação e administrativas, no *rollback* incremental para economia de memória, propostas para *lookahead* dinâmico, entre outros (Capítulo 2). A incorporação destas alternativas ao DCB, vistas como perspectivas futuras para este trabalho, potencialmente podem contribuir com a redução do tempo de simulação.

7 CONCLUSÕES

Esta tese discutiu o uso da simulação heterogênea distribuída como instrumento de validação de sistemas que combinam partes distintas em termos de tecnologia de construção, linguagem de descrição, ou nível de abstração. Neste trabalho, maior atenção foi dedicada aos sistemas que combinam partes de hardware e partes de software (sistemas embarcados). Na bibliografia, a simulação de sistemas embarcados é denominada de co-simulação.

No contexto atual dos desafios no campo da simulação de sistemas heterogêneos, esta tese apresentou, inicialmente, o trabalho de definição e especificação do DCB. O DCB apresenta uma estrutura de suporte à simulação heterogênea distribuída que observa, de modo integrado, a distribuição física de componentes, a independência dos componentes, o encapsulamento das estratégias de gerenciamento de tempo, de dados e de comunicação e a sincronização híbrida. A Seção 3.7 (Capítulo 3) desta tese discute em detalhes as contribuições do DCB nestes vários aspectos.

Em geral, considerando individualmente cada um desses fatores, contribuições com propósitos similares são encontradas principalmente em publicações mais recentes. Contudo, a tendência dos trabalhos em oferecer contribuições em um desses fatores normalmente ocorre em detrimento dos demais. Esta tendência pode ser parcialmente justificada pelo direcionamento de estudos a propósitos limitados por uma área de interesse de uma determinada comunidade de pesquisadores.

Por exemplo, proporcionalmente ao aumento da quantidade de detalhes incorporados a um modelo (redução da abstração) há uma tendência de que a parte de gerenciamento de comunicação entre componentes, locais ou remotos, seja sobrecarregada. Ou seja, quanto maior a proximidade de um modelo da implementação real do sistema que ele representa maior a sua fidelidade, contudo isso pode trazer transtornos como por exemplo a queda de desempenho. Isso não significa que um modelo tenha reduzida sua utilidade, embora níveis mínimos das propriedades de fidelidade e desempenho precisam ser mantidos.

Por outro lado, pesquisas desenvolvidas pela comunidade de simulação têm a ‘simulação’ como atividade fim e geralmente não se detêm em detalhes específicos ou particulares aos domínios que poderiam ser caracterizados como usuários da simulação. Neste contexto, embora a simulação heterogênea esteja sendo mais utilizada, ainda se encontra em processo de consolidação nos estudos onde a simulação é atividade fim, como na área de sistemas embarcados.

Limitando a abrangência desta abordagem para os domínios da simulação de sistemas embarcados, esta realidade faz com que os avanços em simulação ainda sejam pouco explorados nas etapas de validação no projeto desses sistemas. Por exemplo, em [KIM2002] é apresentada uma técnica baseada na redução da freqüência das mensagens

entre simuladores de um modelo com propósitos de aumento de desempenho em co-simulação. No entanto, trabalhos que estudam a redução de mensagens visando desempenho já publicados pela comunidade de simulação distribuída não são citados no artigo. A maior parte da bibliografia cita trabalhos sobre co-simulação publicados em eventos voltados para sistemas embarcados. Esta afirmação em particular não tem como propósito discutir questões de mérito, mas sim argumentar sobre o fato de que ainda é insuficiente a intersecção entre contribuições das duas áreas (sistemas embarcados e simulação distribuída), cuja cooperação traria maiores benefícios para a evolução da co-simulação. Esta característica não é exclusiva da relação entre simulação e sistemas embarcados.

Em geral, no domínio de sistemas embarcados, os estudos fazem uso da simulação para validação seguida de síntese [MCF88]. Por outro lado, no domínio das pesquisas em simulação, embora algumas considerações sejam feitas sobre reuso e interoperabilidade de simuladores distintos, ainda há espaço amplo para amadurecimento deste tema. Por isso, maior esforço pode ser dispendido na direção de uma linha de estudos cuja ênfase seja a 'simulação de sistemas heterogêneos'. O DCB preenche esta lacuna na medida em que incorpora o estado da arte da simulação distribuída e concentra as contribuições no escopo do gerenciamento de modelos heterogêneos distribuídos, abordando com maior ênfase nesta tese a co-simulação de sistemas embarcados. Este esforço faz parte das contribuições do DCB, indo ao encontro das contribuições específicas, como discutidas na Seção 3.7.

No escopo da arquitetura do DCB, esta tese também discutiu requisitos para a construção de um ambiente de modelagem para sistemas heterogêneos, denominado Tangram, cujo desenvolvimento está em andamento no contexto de um trabalho em nível de mestrado. Tangram e DCB visam o projeto e execução de federações heterogêneas, limitando as restrições para composição de novos federados a aspectos de interface. Mais do que isso, o DCB suporta a distribuição física dos federados de forma transparente uma vez que os federados não precisam executar nenhum tipo de operação remota.

O protótipo do DCB apresentado nesta tese permite a geração automática dos módulos de interface para comunicação entre federados e DCB (*gateways*), a geração automática dos arquivos de configuração (XML), e a execução distribuída de federações com o uso do tempo para sincronização de eventos. O protótipo atual não implementa gerenciamento de propriedade e gerenciamento assíncrono do tempo.

Foram construídos quatro experimentos distintos com modelos do SistemaGPSAlerta utilizado como estudo de caso. A cada experimento um novo modelo foi construído com o propósito de impor alterações nos níveis de abstração dos federados e na heterogeneidade. Foram integrados, em diferentes combinações, federados (que representam módulos do SistemaGPSAlerta) descritos em Java, em C e em SystemC. Por exemplo, na transição do experimento 2 para o experimento 3 a interface entre o federado GPSAlerta e o federado Simulador GPS se manteve no mesmo nível de abstração enquanto que a interface do GPS Alerta com o display foi alterada em função da inclusão de um novo federado, o Driver do Display, descrito em C++.

A partir dos experimentos foram criadas 5 versões diferenciadas de federações que representam o SistemasGPSAlerta para coleta de dados com propósitos de medidas de desempenho na execução local e distribuída. Particularmente na execução distribuída, observa-se uma redução expressiva do consumo de tempo na execução de

federações promovido pela política de sincronização no tempo implementada pelo DCB. Os resultados das medidas foram apresentados na Seção 6.4.

7.1 Perspectivas

A continuidade do desenvolvimento do protótipo do Tangram/DCB, com perspectivas à concepção de uma ferramenta completa, pode ser vista como a primeira atividade para a continuidade deste trabalho de tese. O protótipo atual envolve apenas um conjunto suficiente de módulos para a realização dos estudos de caso, como apresentados no Capítulo 6. Contudo, a especificação do DCB e o levantamento de requisitos para o Tangram prevêm novas funcionalidades. Entre elas:

- Suporte à busca, seleção e catalogação de componentes;
- Gerenciamento de tempo no modo assíncrono;
- Alternativas para *breakpoints* em tempo de execução;
- Implementação do gerenciamento de propriedade sobre atributos;
- Implementação de técnicas de verificação da correção de modelos antes do exercício de simulações;
- Módulos para monitoramento de simulações;
- Concepção de uma estratégia mais intuitiva de disparo de federados sem comprometimento das ações de sincronização;
- Aperfeiçoamentos, a partir do estado da arte de pesquisas no campo da simulação distribuída, visando melhoria de desempenho.

As características do DCB permitem que ele seja utilizado como base para a implementação de ambientes de simulação distribuída (heterogênea ou não) para aplicações em diferentes domínios. Contudo, diferentes domínios podem possuir requisitos distintos. Por exemplo, um ambiente de treinamento humano baseado em simulação, se comparado a um ambiente de simulação de componentes eletrônicos, apresenta situações distintas que podem exigir variações em detalhes de implementação do DCB, embora mantendo os princípios definidos em sua arquitetura. Deste modo, o DCB pode ser utilizado para execução de simulações em outros domínios de aplicação. Nesta perspectiva, as pesquisas baseadas no DCB estão no escopo de estudos onde a simulação é atividade meio.

Também faz parte das perspectivas futuras a inclusão de políticas de segurança para os dados que são trocados entre federados e para manter a consistência durante a execução de federações (tolerância a falhas). Além disso, pode-se agregar ao DCB contribuições de estudos em simulação distribuída que visam melhoramentos em questões de performance. Nesta perspectiva, as pesquisas baseadas no DCB estão no escopo de estudos onde a simulação é vista como atividade fim.

REFERÊNCIAS

[1516] INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERING. IEEE 1516-2000: Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules 2000. Disponível em: http://shop.ieee.org/store. Acesso em: 2002.

[ADA96] ADAMS, J. K.; THOMAS, D. E. The Design of Mixed Hardware/Software Systems. In: DESIGN AUTOMATION CONFERENCE, 33.,1996, Las Vegas, Nevada. **Proceedings…** [S.l.]:IEEE Computer Society, 1996. p.515-520.

[BEC99] BECKER, L. B.; PEREIRA, C. E. SIMOO-RT: An Integrated Object-Oriented Environment for the Development of Distributed Real-Time Systems. In: CARS & FOF – CAD/CAM, ROBOTICS & FACTORIES OF THE FUTURE, 1999. **Proceedings...**, Águas de Lindóia, São Paulo: [s.n.], 1999. p.146-149.

[BER2002] BERGAMASCHI, R. A. The A to Z of SoCs**.** In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2002, San Jose, California. **Proceedings…** New York: ACM Press, 2002. p.790-798.

[BIS97] BISHOP, W. D.; LOUCKS, W. M. A Heterogeneous Environment for Hardware/Software Cosimulation. In: 30TH SIMULATION SYMPOSIUM (SS'97), Atlanta, GA, USA, 1997. **Proceedings…** [S.l.:s.n.], 1997. p.14-22.

[BOR2000] BORRIELLO, G; WANT, R. Embedded computation meets the World Wide Web. **Communications of the ACM**, New York, v. 43, p. 59-66, 2000.

[BUS98] BUSS, A.; JACKSON, L. Distributed Simulation Modeling: A Comparison of HLA, CORBA, and RMI. In: WINTER SIMULATION CONFERENCE, 1998, Washington DC, USA. **Proceedings...** [S.l.:s.n.], 1998. p. 819-826.

[CAD2003] CADENCE Incisive™ unified Simulator. Disponível em: <http://www.cadence.com/products/functional_ver/incisive_unified_simulator/index.aspx>. Acesso em: 2003.

[CAI2003] CAI, L.; DANIEL, G. Transaction Level Modeling: An Overview. In: INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODESIGN & SYSTEM SYNTHESIS, 2003, Newport Beach, Califórnia. **Proceedings…** [S.l:s.n.], 2003. p.19-24.

[CAR2000] CARRO, L. et al. A Design Methodology for Embedded Systems based on Multiple Processors. In: DIPES' – IFIP WG10.3/WG10.5 INTERNATIONAL WORKSHOP ON DISTRIBUTED AND PARALLEL EMBEDDED SYSTEMS, 2000, Paderborn, Germany. **Proceedings…** [S.l:s.n.], 2000. p.33-42.

[CAR97] CAROTHERS, C. D. et al. Design and implementation of HLA Time Management in the RTI version F.0. In: WINTER SIMULATION CONFERENCE, 1997, Atlanta, USA. **Proceedings…** [S.l.:s.n.], 1997. p.373-380.

[CES2002] CESÁRIO, W. et al. Component-Based Design Approach for Multicore SoC's**.** In: DESIGN AUTOMATION CONFERENCE, 39., 2002, New Orleans, Louisiana. **Proceedings…** New York: ACM Press, 2002. p.789-794.

[CHA2001] CHATELAIN, A. et al. High-Level Architectural Co-simulation Using Esterel and C. In: INTERNATIONAL SYMPOSIUM ON HARDWARE/SOFTWARE CODESIGN, 9., 2001, Copenhagen, Denmark. **Proceedings…** New York: ACM Press, 2001. p.189-194.

[CHE2001] CHEN, G.; SZYMANSKI, B. K. Component-based simulation. In: EUROPEAN SIMULATION MULTICONFERENCE, ESM, 15., 2001, CTU Prague, Czech Republic. **Proceedings…** [S.l.:s.n.], 2001. p.68-75.

[CHE2001b] CHEN, G.; SZYMANSKI, B. K. Component-oriented simulation architecture: toward interoperability and interchangeability. In: WINTER SIMULATION CONFERENCE, 2001, Arlington, USA. **Proceedings…** [S.l.:s.n.], 2001. p.495-501.

[CHE98] CHETLUR, M. **Reducing Communication Overhead in Asynchronous Distributed Applications.** 1998. 109 f. Master's thesis, Division of Research and Advanced Studies of the University of Cincinnati, Cincinnati, Ohio.

[CHO94] CHOW, A. C.; ZEIGLER, B. P. Parallel DEVS: a Parallel, Hierarchical, Modular Modeling Formalism. In: WINTER SIMULATION CONFERENCE, 1994, Lake Buena Vista, FL. **Proceedings…** [S.l.:s.n.], 1994. p.716-722.

[CHO98a] CHOU, P.; BORRIELLO, G. An Analysis-Based Approach to Composition of Distributed Embedded Systems. In: INTERNATIONAL WORKSHOP ON HARDWARE/SOFTWARE CODESIGN, 1998, Seattle, USA. **Proceedings…** [S.l.:s.n.], 1998. p.3-7.

[CHO98b] CHOU, P.; BORRIELLO, G. Modal Process: Towards Enhanced Retargetability through Control Composition of Distributed Embedded Systems. In: DESIGN AUTOMATION CONFERENCE, 35., 1998, San Francisco, USA. **Proceedings...** [S.l.]: ACM Press, 1998. p.88-93.

[CHO99] CHOU, P. Et al. IPCHINOOK: an integrated IP-based design framework for distributed embedded systems. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION CONFERENCE, 36., 1999, New Orleans, USA. **Proceedings…** New York: ACM Press, 1999. p.44-49.

[COP97] COPSTEIN, B**. SIMOO:** Plataforma Orientada a Objetos para Simulação Discreta Multi-Paradigma. 1997. 137 f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[COS2000] COSTA, J. C.; DEVADAS, S.; MONTEIRO, J. C. Observability Analysis of Embedded Software for Coverage-Directed Validation. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2000, San Jose, California. **Proceedings…** [S.l.:s.n.], 2000. p.27-32.

[COU98] COUMERI, S.L.; THOMAS, D. A simulation environment for hardware-software codesign. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, 1995, Austin, Texas. **Proceedings...** [S.l.:s.n.], 1995. p.58-63.

[COW2003] COWARE. Disponível em: <http://www.coware.com/cowareN2C.html>. Acesso em: 2003.

[CRO95] CROW, J. et al. A tutorial introduction to PVS**.** In: WIFT, 1995, Boca Raton, Florida. **Proceedings…** [S.l.:s.n.], 1995.

[DAH97] DAHMANN, J. S. High Level Architecture for Simulation. In: INTERNATIONAL WORKSHOP ON DISTRIBUTED INTERACTIVE SIMULATION AND REAL-TIME APPLICATIONS-DIS-RT, 1., 1997, Eilat, Israel. **Proceedings…** [S.l.:s.n.], 1997. p.9-14.

[DAH98] DAHMANN, J. S.; FUJIMOTO, R. M.; WEATHERLY, R. M. The DoD High Level Archetecture: An Update. In: WINTER SIMULATION CONFERENCE, 30., 1998, Washington, D.C. **Proceedings…** [S.l.:s.n.], 1998. p.797-804.

[DAL99] DALPASSO, M.; BOGLIOLO, A.; BENINI, L. Virtual simulation of distributed IP-based designs. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION CONFERENCE, 36., 1999, New Orleans, USA. **Proceedings…** [S.l.:s.n.], 1999. p.50-55.

[DAL2000] DALPASSO, M.; BOGLIOLO, A.; BENINI, L. Hardware/software IP protection. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION CONFERENCE, 37., 2000, Los Angeles, California. **Proceedings…** [S.l.:s.n.], 2000. p.593-596.

[DIT98] DITZE, C. et al. WEB-supported Engineering of Computer-based Systems. In: INT. CONFERENCE AND WORKSHOP ON ENGINEERING OF COMPUTER BASED SYSTEMS, 1998, Jerusalen, Israel. **Proceedings…** [S.l.:s.n.], 1998.

[DMS2004] DMSO-Defense and Modeling Simulation Office. [S.l.]: United States Department of Defense. Disponível em: https://www.dmso.mil/public/. Acesso em: 2004.

[DOD2003] DoD Interpretations of the IEEE 1516-2000 series of standards. [S.l.]: United States Department of Defense. Disponível em: <https://www.dmso.mil/public/library/projects/hla/rti/DoD_interps_1516_Release_2.doc>. Acesso em: 2003.

[EFH91] ERNBERG, P. et al. **Guidlines for Specification and Verification of Communication Protocols**. Kista, Sweden.: Swedish Institute of Computer Science, 1991.

[ELN2002] ELNOZAHY, M. et al. Survey of Rollback-Recovery Protocols in Message-Passing Systems**. ACM Computing Surveys (CSUR)**, New York, v.34, n.3, p.375-408, Sept. 2002.

[ELN99] ELNOZAHY, M. et al. **A Survey of Rollback-Recovery Protocols in Message-Passing Systems.** [S.l.]:Carnegie Mellon University, Department of Computer Science, 1999. 45 f. **(**Technical Report CMUCS99148).

[ERN2000] ERNST, R.; JERRAYA, A. A. Embedded system design with multiple languages. In: CONFERENCE ON ASIA SOUTH PACIFIC DESIGN AUTOMATION, 2000, Yokohama, Japan. **Proceedings…** [S.l.:s.n.], 2000. p.391-396.

[FAU99] FAULHABER, N.; SEEPOLD, R. A Flexible Classification Model for Reuse of Virtual Components. **Reuse Techniques for VLSI Design**. Boston: Kluwer Academic, 1999. p.21-36.

[FER95] FERSCHA, A. **Parallel and Distributed Simulation of Discrete Event Systems.** [S.l.]:University of Vienna, Áustria, 1995. 65 f. (Technical Report on Parallel and Distributed Computing).

[FOW2000] FOWLER, M.; SCOTT, K. **UML Essencial.** 2.ed. Porto Alegre: Bookman, 2000. 169 p.

[FRA97] FRANKS, S. et al. State Saving for Interactive Optimistic Simulation. In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 1997, Áustria. **Proceedings…** [S.l.:s.n.], 1997. p.72-79.

[FRE2000] FREY, P. et al. Parallel Mixed-Technology Simulation**.** In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION PADS, 14., 2000. **Proceedings…** [S.l.:s.n.], 2000. p. 7-14.

[FUJ2000] FUJIMOTO, R. M. **Parallel and distributed simulation systems.** [S.l.]: Wiley-Interscience Publication, 2000. 299 p.

[FUJ2001] FUJIMOTO, R. M. Parallel and Distributes Simulation Systems. In: WINTER SIMULATION CONFERENCE, 33., 2001, Arlington, Virginia. **Proceedings…** [S.l.:s.n.], 2001. p.147-157.

[FUJ90] FUJIMOTO, R. M. Parallel Discrete Event Simulation. **Communications of the ACM**, New York, v. 33, n. 10, p.30-53, 1990.

[FUJ92] FUJIMOTO, R. M. State of the art in Parallel Simulation. In: WINTER SIMULATION CONFERENCE, 1992. **Proceedings…** [S.l.:s.n.], 1992. p.246-254.

[FUJ95] FUJIMOTO, R. M. Parallel and Distributed Simulation. In: WINTER SIMULATION CONFERENCE, 1995, Arlington, USA. **Proceedings…** [S.l.:s.n.], 1995. p.118-125.

[FUJ96] FUJIMOTO, R. M.; WEATHERLY, R. Time Management in the DoD High Level Architecture. In: SYMPOSIUM ON PARALLEL AND DISTRIBUTED TOOLS, 1969. **Proceedings...** [S.l.:s.n.], 1996. p. 60-67.

[FUJ99] FUJIMOTO, R. M. Parallel and Distributed Simulation. In: WINTER SIMULATION CONFERENCE, 1999, Phoenix, Arizona. **Proceedings…** New York: ACM Press, 1999. p.122-131.

[FUL96] FULLFORD, D. Distributed Interactives Simulation: It's Past, Present, and Future. In: WINTER SIMULATION CONFERENCE, 1996. **Proceedings…** [S.l.:s.n.], 1996. p.179-185.

[GER2002] GERSTLAUER, A.; GAJSKI, D. D. System-Level Abstraction Semantics. In: INTERNATIONAL SYMPOSIUM ON SYSTEM SYNTHESIS, 15., 2002, Kyoto, Japan. **Proceedings…** [S.l.:s.n.], 2002. p.231-236.

[GHA2002] GHARSALLI, F. et al. Automatic Generation of Embedded Memory Wrapper for Multiprocessor SoC. In: DESIGN AUTOMATION CONFERENCE, 39., 2002, New Orleans, Louisiana. **Proceedings…** [S.l.:s.n.], 2002. p.596-601.

[GOM96] GOMES, F. **Optimizing Incremental State Saving and Restoration.** 1996. 205 f. Ph.D Thesis, Department of Computer Science, University of Calgary, Calgary.

[GOM97] GOMES, F. et al. Multiplexed State Saving for Bounded Rollback. In: WINTER SIMULATION CONFERENCE, 29., 1997, Atlanta, Georgia. **Proceedings…** [S.l.:s.n.], 1997. p.460-467.

[GRA2003] GRANOWETTER, L. RTI Interoperability Issues – API Standards, Wire Standards, and RTI Bridges. In: EUROPEAN SIMULATION INTEROPERABILITY WORKSHOP, 2003, Stockholm, Sweden. **Proceedings...** [S.l.:s.n.], 2003.

[HEI97] HEIM, J. A. Integrating Distributed Simulation Objects. In: WINTER SIMULATION CONFERENCE, 1997. **Proceedings…** [S.l.:s.n.], 1997. p.532-538.

[HER2001] HERZOG, R.; MULDER, J.; ÓXOIS, J. M. In: EUROPEAN SIMULATION INTEROPERABILITY WORKSHOP, 2001, London, UK. **Proceedings…** [S.l.:s.n.], 2001.

[HES2000] HESSEL, F. et al. Interlanguage Communication Synthesis for Heterogeneous Specifications. **Design Automation for Embedded Systems Journal,** [S.l.], v.5, n3, p.223-236, 2000.

[HES2001] HESSEL, F. **Conception des systemes heterogenes multilangages**. 2001. 124 f. These pour obtenir le grade de Docteur, Universite Joseph Fourier, Grenoble, France.

[HES98] HESSEL, F. et al. MCI: Multilanguage Distributed Co-simulação tool. In: WORKSHOP ON DISTRIBUTED AND PARALLEL EMBEDDED SYSTEMS-DIPES, 1998, Paderborn, Germany. **Proceedings…** [S.l.:s.n.], 1998.

[HES99] HESSEL, F. et al. Communication Interface Synthesis for Multilanguage Specifications. In: RAPID SYSTEM PROTORYPING, 1999, Clearwater, Florida. **Proceedings…** [S.l.:s.n.], 1999. p.15-20.

[HIN97a] HINES, K.; BORRIELLO, G. Dynamic communication models in embedded system co-simulation. In: ANNUAL CONFERENCE OF DESIGN AUTOMATION CONFERENCE, 34., 1997, Anaheim. **Proceedings…** [S.l.:s.n.], 1997. p.395-400.

[HIN97b] HINES, K. **Pia:** A Framework For Embedded System Co-simulation with Dynamic Communication Support. [S.l.:s.n.], 1997. (Technical Report UW-CSE-96-11-04).

[HIN97c] HINES, K.; BORRIELLO, G. Selective Focus as a Means of Improving Geographically Distributed Embedded System Co-simulation. In: IEEE INTERNATIONAL WORKSHOP ON RAPID SYSTEM PROTOTYPING, 8., 1997. **Proceedings…** [S.l.:s.n.], 1997. p.58-62.

[HIN98] HINES, K.; BORRIELLO, G. A Geographically Distributed Framework for Embedded System Design and Validation. In: DESIGN AUTOMATION CONFERENCE, 1998. **Proceedings...** [S.l.:s.n.], 1998. p.140-145.

[HOF2001] HOFFMANN, A.; KOGEL, T.; MEYR, H.. A Framework for Fast Hardware-Software Co-simulation. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 2001, Munich, Germany. **Proceedings…** [S.l.:s.n.], 2001. p.760-765.

[HON95] HONEKAMP, U.; STOLPE, R. Design and Application of a Distributed Simulation - and Runtime - Platform for Mechatronic Systems in the Field of Robot Control. In: CONFERENCE ON MECHATRONICS AND ROBOTICS, 3., 1995. **Proceedings…** [S.l.:s.n.], 1995.

[HUB98] HUBERT, H. **A Survey of HW/SW Cosimulaton Techniques and Tools**. 1998. 48 f. Thesis Work, Electronic Systems Design Laboratory, Royal Institute of Technology, Kista, Sweden.

[IND2003] INDRUSIAK, L. S. **A Framework Supporting Collaboration on the Distributed Design of Integrated Systems**. 2003. 179 f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[ITO2001] ITO, S. A.; CARRO, L.; JACOBI, R. P. Making Java Work for Microcontroller Applications. **IEEE Design & Test of Computers,** [S.l.], v.18, n.5. p.100-110, 2001.

[JUN2001] JUNG, J.; YOO, S.; CHOI, K.. Performance Improvement of Multi-Processor Systems Cosimulation based on SW Analysis. In: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 2001, Munich, Germany. **Proceedings…** [S.l.:s.n.], 2001. p.749-756.

[KEA2002] KEATING, M.; BRICAUD, P. **Reuse Methodology Manual for System-on-Chip Designs.** 3.ed. [S.l.]: Kluwer Academic Publishers, 2002.

[KEU2000] KEUTZER, K et al. A. System Level Design: Orthogonolization of Concerns and Platform-Based Design. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, [S.l.], v.19, n.12, Dec. 2000.

[KIM96] KIM, K. et al. An Integrated Hardware-Software Cosimulation Environment with Automated Interface Generation. In: INTERNATIONAL WORKSHOP ON RAPID SYSTEMS PROTOTYPING, 7., 1996. **Proceedings…** [S.l.:s.n.], 1996.

[KIM2002] KIM, D. et al. Virtual Synchronization for Fast Distributed Cosimulation of Dataflow Task Graphs. In: INTERNATIONAL SYMPOSIUM ON SYSTEM SYNTHESIS, 15., 2002, Kyoto, Japan. **Proceedings…** [S.l.:s.n.], 2002. p.174-179.

[KIR94] KIRKWOOD, C. E. **Verification of LOTOS Specifications using Term Rewriting Techiniques.** 1994. Thesis for Degree of Doctor of Philosophy, Department of Computing Science, University of Glasgow, Glasgow.

[KUH2000] KUHL, F.; WEATHERLY, R.; DAHMANN, J. **Creating Computer Simulation Systems:** An Introcuction to the High Level Architecture. [S.l.]: Prentice Hall PTR: The MITRE Corporation, 2000. 212 p.

[LAM78] LAMPORT, L. Time, Clocks, and the Ordering of Events in a Distributed System. **Communications of the ACM**, New York, v.21, n.7. p.558-565, July 1978.

[LAW91] LAW, A. M.; KELTON, W. D. **Simulation Modeling & Analysis.** 2nd ed. New York: McGraw-Hill, 1991. 759 p.

[LEE2000] LEE, E A. What's Ahead for Embedded Software? **IEEE Computer Magazine**, [S.l.], v.33, n.9, p.18-26, 2000.

[LEE2001a] LEE, E. A. et al. **Overview of the Ptolemy Project.** Berkeley: University of California, 2001. 23p. (Technical Memorandum UCB/ERL M01/11).

[LEE2001b] LEE, E. A. Computing for Embedded Systems. In: IEEE INSTRUMENTATION AND MEASUREMENT TECNHOLOGY CONFERENCE, 2001, Budapest, Hungary. **Proceedings…** [S.l.:s.n.], 2001.

[LEE2002] LEE, J.; PARK, I. Timed Compiled-Code Simulation of Embedded Software for Performance Analysis of SOC Design. In: CONFERENCE ON DESIGN AUTOMATION, 39., 2002, New Orleans, Louisiana. **Proceedings…** [S.l.:s.n.], 2002. p.293-298.

[LIA99] LIAO, C.; MARTONOSI, M.; CLARK, D. W. Experience with an adaptative globally-synchronizing clock algorithm. In: ANNUAL ACM SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTURES, 1999, Saint Malo, France. **Proceedings…** [S.l.:s.n.], 1999. p.106-114.

[LIS2002] LISBOA, C. A. L. **Sistema GPSAlerta – Relatório da Implementação**. Porto Alegre: Universidade Federal do Rio Grande do Sul, 2002. (Relatório técnico).

[MCF88] MCFARLAND, M. C.; PARKER, A. C.; CAMPOSANO, R. Tutorial on high-level synthesis. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION CONFERENCE, 25., 1988, Atlantic City, New Jersey. **Proceedings…** [S.l.:s.n.], 1988. p.330-336.

[MEL2001a] MELLO, B. A.; WAGNER, F. R. A Standardized Co-simulation Backbone. In: IFIP INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, 11., 2001, Montpellier. **The Global System on Chip Design & CAD Conference:** Proceedings. Montpellier: LIRMM, 2001. p11-16.

[MEL2001b] MELLO, B. A **Co-simulação Distribuída no suporte à Validação de Sistemas Heterogêneos**. 2001. 99p. Exame de Qualificação (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[MEL2002] MELLO, B. A.; WAGNER, F. R. A Standardized Co-simulation Backbone. In: ROBERT M. et al. (Ed). **SOC Design Methodologies.** Boston: Kluwer Academic, 2002. p.181-192.

[NAG2002] NAGENDRA, G. D.; KUMAR, V. G. P.; SHESHADRI, B. S. S. Simulation Bridge: A Framework for multi-processor simulation. In: INTERNATIONAL SYMPOSIUM ON HARDWARE/SOFTWARE CODESIGN, 10., 2002, Easte Park, Colorado. **Proceedings…** [S.l.:s.n.], 2002. p.49-54.

[NAN99] NANCE, E. R. Distributed Simulation with Federated Models: Expectations, Realizations and Limitations. In: WINTER SIMULATION CONFERENCE, 1999. **Proceedings...** [S.l.:s.n.], 1999. p.1026-1031.

[NIC2001] NICOLESCU, G.; YOO, S.; JERRAYA, A. Mixed-level cosimulation for fine gradual refinement of communication in SoC design. In: DATE, 2001, Munich, Germany. **Proceedings...** [S.l.:s.n.], 2001. p.754-759.

[NIC2002] NICOLESCU, G. Application of Multi-domain and Multi-language Cosimulation to an Optical MEM Switch Design. In: INTERNATIONAL CONFERENCE ON VLSI DESIGN, VLSI, 15., 2003. **Proceedings…** [S.l.:s.n.], 2003.

[NOW99] NOW, Y-H. et al. Survey of Languages and Runtime Libraries for Parallel Discrete-Event Simulation. **Simulation**, [S.l.], p.170-184, 1999.

[NYG81] NYGAARD, K.; DAHL, O-J. The Development of the SIMULA Languages. In: WEXELBLAT R. L. (Ed). **History of Programming Languages.** New York: Academic Press, 1981. p. 439-493.

[OMN2004] OMni – FOM independence with the Standard HLA API. Disponível em: <http://www.aegistg.com/labcut/lwproducts/omni/omni.htm>. Acesso em: 2004.

[ORT98a] ORTEGA, R. B.; BORRIELLO, G. Communication Synthesis for Distributed Embedded Systems. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1998, California. **Proceedings…** [S.l.:s.n.], 1998. p. 437-444.

[ORT98b] ORTEGA, R. B.; LAVAGNO, L.; BORRIELLO, G. **Models and Methods for HW/SW Intellectual Property Interfacing.** NATO-Advanced Study Institute, 1998, 37 p.

[OYA2000] OYAMADA, M.; WAGNER, F. R. Co-simulation of Embedded Electronic Systems. In: EUROPEAN SIMULATION SYMPOSIUM, 12., 2000, Hamburg. **Proceedings…** Ghent:SCS, 2000. p630-634.

[PAG2000] PAGE, E. H. et al. Web-Based Simulation: Revolution or Evolution? **ACM Transactions on Modeling and Computer Simulation**, [S.l.], v.10, n.1, p.3-17, Jan. 2000.

[PAG99] PAGE, E. H. Beyond speedup: PADS, the HLA and Web-based Simulation. In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 13., 1999, Atlanta. **Proceedings…** [S.l.:s.n.], 1999.

[PAD2000] PANIGRAHI, D.; TAYLOR, C. N.; DEY, S. Interface Based Hardware/Software Validation of a System-on-Chip. In: IEEE INTERNATIONAL HIGH-LEVEL VALIDATION AND TEST WORKSHOP, 2000, Berkeley, California. **Proceedings…** [S.l.:s.n.], 2000. p.53-58.

[PAN2001] PANDA, P. R. SystemC – A modeling platform supporting multiple design abstractions. In: INTERNATIONAL SYMPOSIUM ON SYSTEMS SYNTHESIS, 14., 2001, Montréal. **Proceedings…** [S.l.:s.n.], 2001. p.75-80.

[PAT2002] PATERAS, S. Embedded Diagnosis IP. In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION DATE, 2002. **Proceedings…** [S.l.:s.n.], 2002. p.242.

[PER2001] PERUMALLA, K.; FUJIMOTO, R. M. Virtual Time Synchronization over Unreliable Network Transport. In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 15., 2001, Lake Arrowhead, California. **Proceedings…** [S.l.:s.n.], 2001. p.129-136.

[PET93] PETERSON, G. D.; CHAMBERLAIN, R. D. Exploiting *Lookahead* in Synchronous Parallel Simulation. In: WINTER SIMULATION CONFERENCE, 25., 1993, Los Angeles, California. **Proceedings…** [S.l.:s.n.], 1993. p.706-712.

[RAD98] RADHAKRISHNAN, R. et al. An Object-Oriented Time Warp Simulation Kernel. In: INTERNATIONAL SYMPOSIUM ON COMPUTING IN OBJECT-ORIENTED PARALLEL ENVIRONMENTS, ISCOPE, 1998. **Proceedings…** [S.l.:s.n.], 1998. p.13-23.

[RAM2000] RAMMIG, F. Web-based System Design with Components off the Shelf. In: FORUM ON DESIGN LANGUAGES, FDL, 2000, Tübingen. **Proceedings...** [S.l.:s.n.], 2000.

[RAO2000a] RAO, D. M.; CHERNYAKHOVSKY, V.; WILSEY, P. A. WESE: A Web-based Environment for Systems Engineering. In: INTERNATIONAL CONFERENCE ON WEB-BASED MODELLING & SIMULATION, WEBSIM, 2000. **Proceedings...** [S.l.:s.n.], 2000.

[RAO2000b] RAO, D. M.; WILSEY, P. A. Dynamic Component Substitution in Web-Based Simulation. In: WINTER SIMULATION CONFERENCE, 2000. **Proceedings...** [S.l.:s.n.], 2000.

[REU99] REUTTER, A.; ROSENSTIEL, W. An Efficient Reuse System for Digital Circuit Design. In: DESING, AUTOMATION AND TEST IN EUROPE, 1999, Munich. **Proceedings**... [S.l.:s.n.], 1999. p. 497-508.

[ROW97] ROWSON, J. A.; SANGIOVANNI-VINCENTELLI, A. Interface-Based Design. In: DESIGN AUTOMATION CONFERENCE, 34., 1997, Anaheim, California. **Proceedings…** [S.l.:s.n.], 1997. p.178-183.

[SAS2004] SASHIMI PROJECT HOME PAGE. **Sashimi**. Disponível em <http://www.inf.ufrgs.br/sashimi/software.html>. Acesso em: fev. de 2004.

[SCH2003] SCHUBERT, K. Improvements in functional simulation addressing challenges in large, distributed industry projects. In: CONFERENCE ON DESIGN AUTOMATION, 40., 2003, Anaheim. **Proceedings…** [S.l.:s.n.], 2003. p. 11-14.

[SEA2003] SEAMLESS CVT. Mentor Graphics. Disponível em: <http://www.mentorg.com/seamless>. Acesso em: 2003.

[SIS2004] SISO-Simulation Interoperability Standards Organization. Disponível em: <http://www.sisostds.org/>. Acesso em: 2004.

[SOU2003] SOUZA, U. R. F.; SPERB, J. K.; MELLO, B. A.; WAGNER, F. R. Tangram – Virtual Integration of Heterogeneous IP Components in a Distributed Co-simulation Environment. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 16., 2003, São Paulo. **Proceedings...** Los Alamitos: IEEE Computer Society, 2003. p.125-130.

[SPE2003] SPERB, J. K. **Geração de Modelos de Co-simulação Distribuída para a Arquitetura do DCB.** 2003. 102 f. Dissertação (Mestrado em Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[SPO2001] SPOLON, R. **Um Método para Avaliação de Desempenho de Protocolos de Sincronização Otimistas para Simulação Distribuída.** 2001. 247 f. Tese (Doutorado) - Departamento de Física e Informática, Universidade de São Paulo, São Paulo.

[STE2003] STEINMAN, J. S.; WONG, J. W. The SPEEDES Persistence Framework and the Standard Simulation Architecture. In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 17., 2003, Washington. **Proceedings…** [S.l.:s.n.], 2003. p.11-20.

[STE95] STEINMAN, J.S. et al. Global virtual time and distributed synchronization. In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 9., 1995, New York. **Proceedings…** [S.l.:s.n.], 1995. p. 139-148.

[STO98] STOPLE, R.; ZANELLA, M. A Distributed Hardware-in-the-Loop Simulation Environment in Use on a Testbed of a Series Hybrid Drive. In: EUROPEN SIMULATION MULTICONFERENCE, 12., 1998, Manhester. **Proceedings…** [S.l.:s.n.], 1998.

[STR98] STRABURGER, S. et al. Internet-based simulation using off-the-shelf simulation tools and HLA. In: WINTER SIMULATION CONFERENCE, 1998, Washington. **Proceedings...** [S.l.:s.n.], 1998. p. 1669-1676.

[STY2001] STYTZ, M. R.; BANKS, S. B. XML in Distributed Simulation for Data Management and Archetecture Description. In: SIMULATION INTEROPERABILITY WORKSHOP SPRING, 2002, Orlando, Florida. **Proceedings…** [S.l.:s.n.], 2002.

[STY2002] STYTZ, M. R. Utilizing the Aspect Oriented Programming in Conjunction with the Unified Modeling Language and Object Oriented Programming for Distributed Simulation. In: SIMULATION INTEROPERABILITY WORKSHOP, 2002, Orlando, Florida. **Proceedings…** [S.l.:s.n.], 2002.

[SUN97] SUNG, W.; OH, M.; HA, S. Interface Design of VHDL Simulation for Hardware-Software Co-simulation. In: ASIA PACIFIC CONFERENCE ON HARDWARE DESCRIPTION LANGUAGES, APCHDL, 1997, Hsin-Chu, Taiwan. **Proceedings…** [S.l.:s.n.], 1997. p.43-49.

[SUN98] SUNG, W.; HA, S. Hardware Software Co-simulação Backplane with Automatic Interface Generation. In: ASP-DAC, 1998. **Proceedings…** New York: ACM SIGDA, 1998.

[SYN2003] SYNOPSYS. Disponível em: <http://www.synopsys.com/products/hwsw/eagle_ds.html>. Acesso em: 2003.

[TAN2003] TANENBAUM, A. S. **Sistemas Operacionais Modernos.** 2.ed. São Paulo: Prentice Hall, 2003. 695 p.

[TAY99] TAYLOR, S. J. E.; SAVILLE, J.; SUDRA, D. M. Developing Interest Management Techniques in Distributed Interactive Simulation Using Java. In: WINTER SIMULATION CONFERENCE, 1999. **Proceedings...** [S.l.:s.n.], 1999. p.518-523.

[TEO2002] TEO, Y. M.; NG, Y. K.; ONGGO, B. S. S. Conservative Simulation using Distributed-Shared Memory. In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION 16., 2002, Washington-D.C. **Proceedings…** [S.l.:s.n.], 2002. p.03-10.

[TOL2002] TOLK, A. Avoiding another Green Elephant – A Proposal for the Next Generation HLA based on the Model Driven Architecture. In: SIMULATION INTEROPERABILITY WORKSHOP, 2002, Orlando, Florida. **Proceedings…** [S.l.:s.n.], 2002.

[VAC99] VACCARO, G. L. R. **Simulação Paralela e Distribuída com vistas ao Co-Design.** 1999. 65 p. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[VHD2000] VHDL Modeling Terminology and Taxonomy. Version 3.1, may/2000. Disponível em: <http://www.atl.external.lmco.com/rassp/taxon/rassp_taxon.html>. Acesso em: ago. 2001.

[VIL2001] VILLELA, C.; BECKER, L.B.; PEREIRA, C.E. Framework for Component-Based Development of Distributed Real-Time Systems. In: WORDS, 2001, Roma. **Proceedings...** [S.l.:s.n.], 2001.

[WAG94] WAGNER. F. R. **Ambientes de Projeto de Sistemas Eletrônicos**. Recife: UFPE, 1994. 156 p.

[WAG96] WAGNER, P. R; FREITAS, C.M.D.S.; WAGNER, F. R. A New Paradigm for Visual Interactive Modeling and Simulation. In: EUROPEAN SIMULATION SYMPOSIUM, 8., 1996, Genova. **Proceedings…** [S.l.]: Society for Computer Simulation, 1996. v.1. p. 142-145.

[WAG99] WAGNER, F. R.; OYAMADA, M.; CARRO, L.; KREUTZ, M. Object-Oriented Modeling and Co-simulation of Embedded Electronic Systems in the S3E2S Environment. In: UFRGS MICROELECTRONICS SEMINDR, 14., 1999, Pelotas. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 1999. p.113-116.

[WAG2004] WAGNER, F. R.; CESARIO, W.O.; CARRO, L.; JERRAYA, A. A., Strategies for the Integration of Hardware and Software IP Components in Embedded System-on-Chip. **Integration VLSI Journal**, [S.l.], v.37, n.4, Sept. 2004.

[WIL99] WILSEY, P. A. Web-Based Analysus and Distributed IP. In: WINTER SIMULATION CONFERENCE, 1999. **Proceedings…** [S.l.:s.n.], 1999. p. 1445-1453.

[WILD2001] WILDT, D.; WAGNER, F.R. Adapting Simulation Environments to HLA: Discussion and Case Study. In: EUROPEAN SIMULATION MULTICONFERENCE, 2001, Prague. **Proceedings…** San Diego:SCS, 2001. p.601-605.

[YI2003] YI, Y.; KIM, D., HA, S. Virtual Synchronization Technique with OS Modeling for Fast and Time-accurate Cosimulation. In: IEEE/ACM/IFIP INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODESIGN, 2003, Newport Beach. **Proceedings…** [S.l.:s.n.], 2003. p.1-6.

[YOO2000] YOO, S.; CHOI, K.; HA, D. S. Performance Improvement of Geographically Distributed Co-simulação by Hierarchically Grouped Messages. **IEEE Transactions on VLSI Systems**, New York, v.8, n.5. p. 100-104, Oct. 2000.

[YOO2001] YOO, S. et al. A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design. In: CODES, 2001, Copenhague, Danemark. **Proceedings…** [S.l.:s.n.], 2001.

[YOO97] YOO, S.; CHOI, K. Optimistic Timed HW-SW Co-simulação. In: ASIA-PACIFIC CONFERENCE ON HARDWARE DESCRIPTION LANGUAGE, 4., 1997. **Proceedings…** [S.l.:s.n.], 1997. p.39-42.

[YOO98] YOO, S.; CHOI, K. Optimistic distributed timed co-simulação based on thread simulation model. In: INTERNATIONAL WORKSHOP ON HARDWARE/SOFTWARE CODESIGN, 6., 1998, Seattle. **Proceedings…** [S.l.:s.n.], 1998. p. 71-75.

[ZAN2000] ZANELLA, M. et al. A. Structuring and Distribution of Controller Software in Dependence of the System Structure. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 13., 2000, Florianópolis. **Anais...** Florianópolis: SBA, 2000.

[ZEI99a] ZEIGLER, B. P.; KIM, D.; BUCKLEY, S. J. Distributed Supply Chain Simulation in a DEVS/CORBA Execution Environment. In: WINTER SIMULATION CONFERENCE, 1999. **Proceedings...** [S.l.:s.n.], 1999. p.1333-1340.

[ZEI99b] ZEIGLER, B. P. et al. Implementation of the DEVS Formalism over the HLA/RTI: Problems and Solutions. In: ADVANCE SIMULATION TECHNOLOGY THRUST, ASTT, 1999, Orlando, Flórida. **Proceedings...** [S.l.:s.n.], 1999.