Um Framework de Apoio à Colaboração no Projeto Distribuído de Sistemas Integrados

Leandro Soares Indrusiak 1,2

Ricardo A. L. Reis 1

Manfred Glesner 2

Resumo: O trabalho de pesquisa apresentado neste artigo tem por objetivo possibilitar a distribuição flexível de recursos e ferramentas de apoio ao projeto de sistemas integrados, considerando especificamente a necessidade de interação colaborativa entre os projetistas. O trabalho enfatiza particularmente alguns problemas que foram considerados apenas marginalmente em abordagens anteriores, como a abstração da distribuição em rede dos recursos de automação de projeto, o controle de consistência na interação síncrona e assíncrona entre projetistas e o suporte a modelos extensíveis de dados de projeto.

Abstract: The work described in this paper aims to allow the flexible distribution of resources and tools supporting the design of integrated systems and considers specifically the need for collaborative interaction among designers. Particular emphasis was given to issues which were only marginally considered in previous approaches, such as the abstraction of the distribution of design automation resources over the network, the consistency control on both synchronous and asynchronous interaction among designers and the support for extensible design data models.

1 Instituto de Informática, UFRGS, Caixa Postal 15064, 91501-970, Porto Alegre, RS, Brasil
{reis@inf.ufrgs.br}
2 Microelectronic Systems Institute, TU Darmstadt, Karlstr. 15, 64283 Darmstadt, Germany
{indrusiak, glesner@mes.tu-darmstadt.de}

1 Introdução

1.1 Contexto

O trabalho de pesquisa apresentado neste artigo tem por objetivo apoiar o projeto de sistemas integrados em ambiente distribuído, considerando especificamente a necessidade de interação colaborativa entre os projetistas. Esta necessidade é claramente destacada em publicações como o SIA Roadmap [1] e o Medea+ Roadmap [2], que analisam os principais problemas enfrentados pela industria de semicondutores e as possíveis soluções advindas de pesquisa nos próximos 10 anos. O presente trabalho enfatiza particularmente alguns problemas que foram considerados apenas marginalmente em abordagens anteriores, como a abstração da distribuição em rede dos recursos de automação de projeto, a possibilidade de interação síncrona e assíncrona entre projetistas e o suporte a modelos extensíveis de dados de projeto.

Tais problemas requerem uma infra-estrutura de software significativamente complexa, pois possíveis soluções envolvem diversos módulos, desde interfaces com o usuário (GUIs) até bancos de dados e *middleware*. Para construir tal infra-estrutura, várias técnicas de engenharia foram empregadas e algumas soluções originais foram desenvolvidas. A idéia central da solução proposta é baseada no emprego conjunto de duas tecnologias homônimas: CAD Frameworks (ambientes integrados de apoio ao projeto) e frameworks orientados a objeto. O primeiro conceito foi criado no final da década de 80 na área de automação de projeto de sistemas eletrônicos e define uma arquitetura de software em níveis, voltada ao apoio a desenvolvedores de ferramentas de projeto, administradores de ambientes de projeto e projetistas. O segundo, desenvolvido na última década na área de engenharia de software, é um modelo para arquiteturas de software visando o desenvolvimento de subsistemas reusáveis de software orientado a objeto. No presente trabalho, propõe-se a criação de um framework orientado a objetos que inclui conjuntos extensíveis de primitivas de dados de projeto bem como de blocos para a construção de ferramentas de CAD. Esse framework orientado a objeto é agregado a um CAD Framework, onde ele passa a desempenhar funções tipicamente encontradas em tal ambiente, tais como representação e administração de dados de projeto, versionamento, interface com usuário, administração de projeto e integração de ferramentas.

1.2 Projeto Colaborativo

A interoperabilidade entre ferramentas de projeto foi um dos tópicos mais importantes da pesquisa cobertos pela área da automatização de projeto de circuitos integrados nos últimos trinta anos. Recentemente, a interoperabilidade entre projetistas passou a receber atenção, e a necessidade de técnicas específicas para suportar a comunicação, a coordenação e o compartilhamento de dados entre grupos de projetistas ficou clara [1,2]. A razão é óbvia:

a complexidade do projeto de circuitos integrados está aumentando mais rápido que o previsto [3].

Observa-se também que o mercado afasta-se lentamente do paradigma baseado em computadores pessoais, passando a um cenário onde os recursos computacionais estão distribuídos entre diversos dispositivos de menor porte. Estes dispositivos oferecem ao usuário possibilidades mais simplificadas de operação – essa é justamente uma das vantagens do novo paradigma – mas por outro lado a complexidade de projeto de tais dispositivos é ainda elevada. Essa complexidade – que pode envolver subsistemas digitais, analógicos, óticos e eletromecânicos, bem como as interfaces de programação de tais subsistemas – só pode ser dominada por grupos de projetistas trabalhando colaborativamente na busca de soluções dos problemas de projeto.

A pesquisa na área de projeto colaborativo pode ser considerada uma combinação dos esforços de pesquisa em CAD (projeto assistido por computador) e CSCW (trabalho colaborativo assistido por computador). Ambas áreas já incorporam uma quantidade significativa de conhecimento. A pesquisa inter-disciplinar envolvendo ambas áreas também já atingiu certa maturidade, especialmente nas áreas de apoio ao projeto de engenharia mecânica e civil. Na área abordada no presente trabalho – projeto de sistemas integrados de hardware e software - a pesquisa em projeto colaborativo é ainda incipiente, e boa parte dos trabalhos disponíveis na literatura foram analisados em [4].

As abordagens visando apoiar projeto colaborativo podem ser caracterizadas de acordo com a taxonomia de tempo e espaço utilizada na área de CSCW [5] (Tabela 1). Considerando o caso onde equipes de projetistas estão distribuídas geograficamente, assumimos que a infraestrutura de apoio ao projeto colaborativo deva contemplar os tipos de colaboração mostrados na segunda linha da Tabela 1.

Tabela 1. Taxonomia tempo-espaço para sistemas CSCW

	mesmo tempo	tempos distintos
mesmo espaço	interação face-a-face	interação assíncrona
espaços distintos	interação síncrona distribuída	interação assíncrona distribuída

Trabalhos anteriores relativos à área de projeto colaborativo contemplavam principalmente processos de colaboração assíncrona. Contribuições de Katz [6], Harrison [7] e Wagner [8] na pesquisa e desenvolvimento de modelos de dados de projeto suportando versões possibilitaram um aumento da eficiência nos casos onde grupos de projetistas trabalham concorrentemente de forma assíncrona. Contribuições em gerência de metodologias de projeto e modelagem de fluxo de projeto – tais como de Brglez [9] e Schneider [10] – também são relevantes, uma vez que permitem que os líderes de grupos de projetistas definam e disponibilizem a suas equipes um conjunto de regras e procedimentos a serem seguidos durante o processo de projeto.

A abordagem descrita no presente trabalho difere dos trabalhos anteriores por (1) prover suporte tanto à colaboração síncrona quanto assíncrona, bem como por (2) definir

explicitamente uma separação de domínios entre o modelo da semântica de projeto e sua representação visual.

A primeira característica é de particular importância para a automação de projeto de sistemas integrados, pois a colaboração síncrona é necessária nas etapas iniciais do projeto enquanto a colaboração assíncrona ocorre nas etapas finais, durante a implementação e verificação do sistema sendo projetado. Por exemplo, um alto potencial de colaboração pode ser identificado nos primeiros passos de processo de projeto, quando a funcionalidade e os requisitos técnicos do produto são definidos, justamente devido a multi-disciplinaridade de tais atividades. Engenheiros de hardware, programadores, gerentes de marketing e de produto são alguns dos profissionais que estariam envolvidos em tais atividades, onde a colaboração síncrona seria indispensável. Por outro lado, durante as etapas de implementação e verificação – desenvolvimento e integração de componentes de hardware, programação, *debugging*, etc. – o potencial de colaboração não deve ser muito elevado, pois desenvolvedores tendem a trabalhar individualmente e de forma assíncrona.

A segunda característica leva em conta as várias possibilidades de entrada e visualização de dados de projeto de sistemas integrados. Descrições gráficas, tais como esquemáticos de circuitos, diagramas de estado e diagramas UML, são usados concorrentemente com descrições textuais na forma de código de linguagens de descrição de hardware ou linguagens de programação, resultando em um cenário onde a colaboração pode ser dificultada pela heterogeneidade entre as formas de modelar e visualizar os dados de projeto. Para minimizar esse problema, o presente trabalho possibilita que projetistas utilizem diferentes formas de entrada e visualização dos dados de projeto, sempre mantendo a consistência entre cada visualização e a semântica do projeto.

A solução proposta no presente trabalho utilizou e ampliou o ambiente de projeto Cave [11], incluindo em sua estrutura um *framework* orientado a objetos que é responsável pela modelagem de dados de projeto e pela instanciação de ferramentas de apoio ao projeto. O suporte ao projeto colaborativo foi incluído nesse *framework*, de forma que futuras extensões ao modelo de dados de projeto ou ao conjunto de ferramentas poderão também utilizar tal recurso. A versão ampliada foi chamada de Cave2, e inclui também um conjunto de serviços que controla o funcionamento das sessões de projeto colaborativo chamado Service Space.

2 Arquitetura proposta

Nos últimos 20 anos, vários grupos se dedicaram à pesquisa na área de automação de projeto utilizando os chamados *frameworks* de CAD. Esses *frameworks* foram criados para definir uma camada entre o sistema operacional das estações de trabalho e os demais softwares que são necessários durante o projeto de um sistema integrado [12]. Dessa forma, toda a atividade de projeto seria realizada através de um ambiente integrado de software de CAD. A funcionalidade de tais *frameworks* inclui gerência de dados, suporte a desenvolvedores de ferramentas, suporte a integração e comunicação entre ferramentas.

Entretanto, as tendências no mercado de automação de projeto de sistemas eletrônicos seguiu o conceito de "best-tool-of-the-class": projetistas preferiram usar ferramentas

individuais, específicas para determinadas atividades e desenvolvidas por diferentes fornecedores, ao invés de adotar a solução completa de um único fornecedor. Essa tendência se deve ao fato de que nenhum fornecedor é capaz de prover a melhor ferramenta para cada etapa do projeto, tal a complexidade de cada uma delas hoje em dia. Essa situação, aliada ao fracasso das iniciativas de padronização de frameworks de CAD integrando soluções de múltiplos fornecedores, resultou na rejeição do conceito de frameworks de CAD sob o ponto de vista comercial. Sob o ponto de vista técnico, a falta de flexibilidade foi o principal motivo da utilização restrita dos frameworks de CAD: como eles dependiam de uma padronização que não ocorreu, tornaram-se praticamente inusáveis por não poder adaptar e evoluir seus modelos de dados de projeto, suas arquiteturas para comunicação entre ferramentas e suas interfaces com o projetista.

Apesar disso, vários grupos de pesquisa chegaram a resultados significativos, e que podem ainda ser usados no suporte à colaboração entre projetistas [7,8,13]. A abordagem proposta leva em conta tais avanços, mas os utiliza dentro de um novo contexto ao fazer uso de técnicas de engenharia de software que não estavam disponíveis quando do desenvolvimento da primeira geração de *frameworks* de CAD.

3.1 Modelagem de dados e de ferramentas usando frameworks orientados a objetos

De acordo com Johnson [14], um *framework* orientado a objetos é um projeto reutilizável de software definido por um conjunto de classes abstratas e pela maneira pela qual as instâncias dessas classes colaboram entre si. Levando em conta tal conceito, exploramos neste trabalho a possibilidade de que um framework de CAD possa incorporar um framework orientado a objetos, aqui definido como um conjunto de classes abstratas que modelam as primitivas de dados de projeto bem como as ferramentas de CAD que as manipulam. Esse framework orientado a objetos reutiliza diversos padrões de projeto, que são soluções já validadas para problemas tipicamente encontrados em arquiteturas complexas de software. A análise detalhada da arquitetura proposta neste trabalho mostra claramente a aplicação dos padrões *Composite*, *Observer*, *Proxy*, *Chain of Responsibility*, *Bridge* e *Façade* [15], entre outros. Ao incluir um framework orientado a objetos, foi possível atingir um nível mais alto de abstração no que tange a funcionalidade de frameworks de CAD: gerência e armazenamento de dados, padrões de comunicação entre ferramentas e suporte a extensibilidade do ambiente de automação de projeto.

O ambiente de projeto é implementado pelas sub-classes concretas derivadas do núcleo de classes abstratas do framework. Esse núcleo é dividido em dois pacotes principais. O primeiro é um conjunto extensível de primitivas de interface gráfica e de automação de projeto, usados para a instanciação de ferramentas de CAD que fazem a interface entre o projetista e o ambiente de projeto. Tais ferramentas são dinamicamente montadas de acordo com as necessidades e objetivos do projetista. O segundo pacote é um conjunto de primitivas de dados de projeto que podem ser instanciadas pelos projetistas à medida em que eles interagem com as ferramentas ao criar um projeto. A Figura 1 mostra como ambos conjuntos de classes interagem com os projetistas durante o processo de projeto. Primeiramente, as ferramentas são requisitadas e executadas através da instanciação e integração das primitivas de interface gráfica e automação de projeto. Uma vez instanciadas,

essas ferramentas passam a construir o projeto ao instanciar as primitivas de dados de projeto sob o comando do projetista.

Para armazenar os dados de projeto ao final de cada sessão de projeto, bem como para permitir acesso multi-usuário aos dados - um requisito para colaboração síncrona - faz-se necessário um conjunto de serviços de persistência e consistência de dados. Figura 2 ilustra como tais serviços apóiam o ambiente de projeto ao permitir o acesso multi-usuário às primitivas de dados instanciadas pelos projetistas. Na presente abordagem, as primitivas de dados de projeto são acessadas a partir de módulos de ferramentas específicos, que são responáveis pela visualização e edição dos dados de projeto, assim como pela comunicação com os mecanismos de controle de consistência

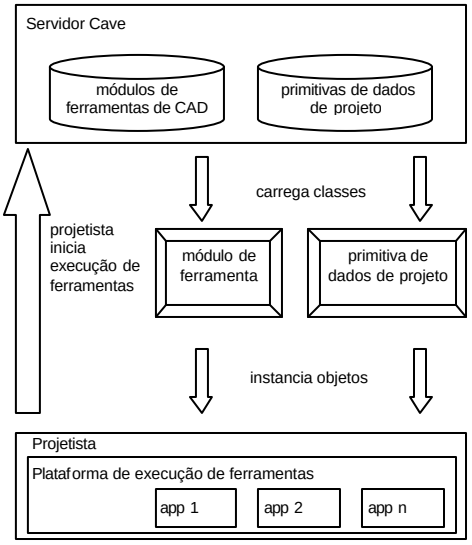

Figura 1. *Frameworks* de dados de projeto e de ferramentas no servidor Cave

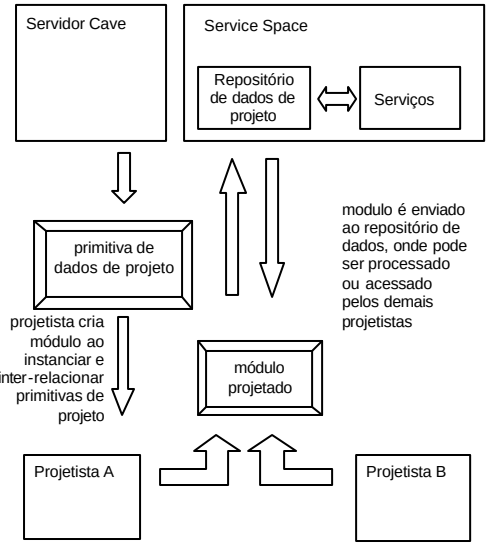

Figura 2. Compartilhamento de dados de projeto através do Service Space

Ao incluir um framework orientado a objetos no núcleo do ambiente Cave, um grau de padronização significativo foi atingido uma vez que as dependências semânticas entre os módulos de ferramentas de CAD e as primitivas de dados de projeto estão definidas no nível abstrato do framework. Ou seja, o framework contém definições genéricas de tipos de dados que permitem às ferramentas compreender que tipos de operações são permitidas, por exemplo composição hierárquica, agregação e instanciação. Além disso, o framework provê mecanismos que garantem que tais operações sejam realizadas de forma consistente em um ambiente multi-usuário usando mecanismos de controle mais ou menos flexíveis, de acordo com a semântica da operação.

Essa padronização, entretanto, não reduziu a flexibilidade do ambiente de projeto, pois sua inerente extensibilidade permite concretizações sucessivas das relações especificadas no nível abstrato. Em outras palavras, os módulos de ferramentas de CAD e as primitivas de dados de projeto são apenas pontos de partida, possibilitando extensões e especializações sem perder a compatibilidade com os serviços e recursos do ambiente de projeto, descritos na próxima subseção. Por exemplo, a operação de composição hierárquica pode ser usada tanto em um esquemático de um circuito de portas lógicas quanto em um diagrama de sequência UML 2.0, e ambos seriam beneficiados pelos mecanismos de controle de consistência.

Na Figura 3, pode-se ver uma parte do modelo de dados expresso no framework de primitivas de dados de projeto.

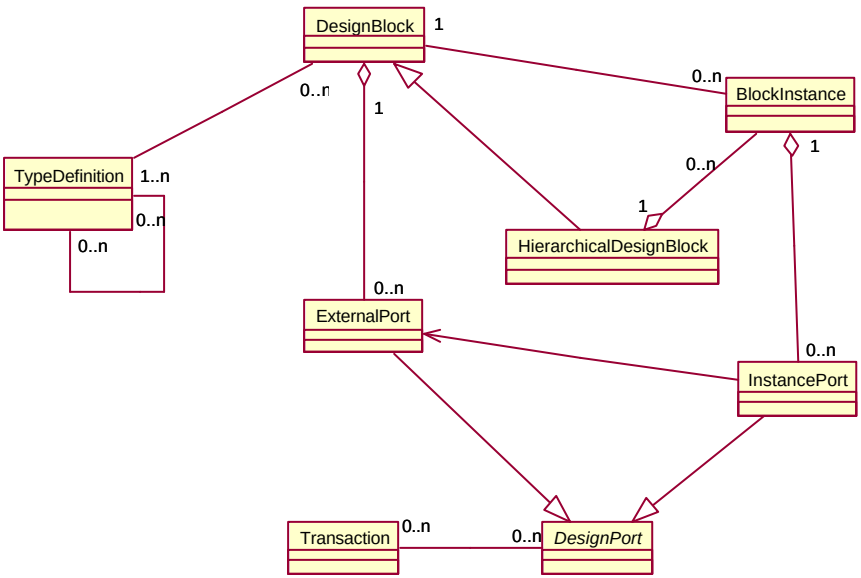

Figura 3. Diagrama de classes mostrando parcialmente o framework de primitivas de dados de projeto

3.2 Integração de serviços

A abordagem aqui proposta inclui um elemento chamado Service Space, mostrado na Figura 2. A funcionalidade do Service Space inclui (1) a integração de ferramentas de CAD externas; (2) um ambiente de execução para os serviços internos, como repositórios de dados, controle de consistência e autenticação de usuários; e (3) a infraestrutura de

localização e disponibilização de serviços, visando possibilitar a inclusão e exclusão dinâmica de serviços.

Existem várias possibilidades para a implementação desse elemento, tais como CORBA, *webservices* baseados em SOAP, Jini, além de outras abordagens não relacionadas com o paradigma de orientação a objetos. A escolha da tecnologia Jini para a implementação do presente protótipo foi baseada em algumas de suas peculiaridades, como o fato dela ser baseada em Java (assim como a implementação do framework orientado a objetos incluso no Cave2), de ter ferramentas de desenvolvimento disponíveis gratuitamente e de incorporar a maioria dos recursos necessários para localização e disponibilização de serviços. Jini também inclui um modelo de programação – desenvolvido sobre as fundações da linguagem Java – de forma a prover ao desenvolvedor as primitivas necessárias para a implementação de cessão de serviços (*leases*), transações e eventos. A chamada remota de métodos também depende de recursos da tecnologia Java, mais especificamente do pacote JavaRMI. Tal modelo de programação usa *proxies* para permitir referências locais a objetos remotos, permitindo que no domínio da aplicação todas as chamadas de métodos sejam feitas a objetos residentes na memória local, abstraindo as dificuldades inerentes à utilização de subsistemas remotos.

O Service Space está acessível através de um protocolo de descoberta (*discovery*), que permite aos clientes utilizar o servidor de *lookup*. Todos os serviços conectados ao Service Space usam a interface *Join* para notificar sua localização na rede e suas características de acesso. Já os clientes usam a interface de *Lookup* para procurar pelos serviços que pretendem utilizar. Os serviços conectados ao Service Space incluem ferramentas de CAD externas assim como serviços internos que são parte do ambiente implementado, como os mecanismos de autenticação de usuários, vários módulos de controle de acesso concorrente (bloqueios, transações, etc.), a interface de acesso ao repositório de dados e os serviços de prototipação. Tanto os serviços internos quanto as ferramentas externas podem ser incluídos dinamicamente, contribuindo assim com a escalabilidade dessa solução. Figura 4 traz uma visão geral dessa abordagem.

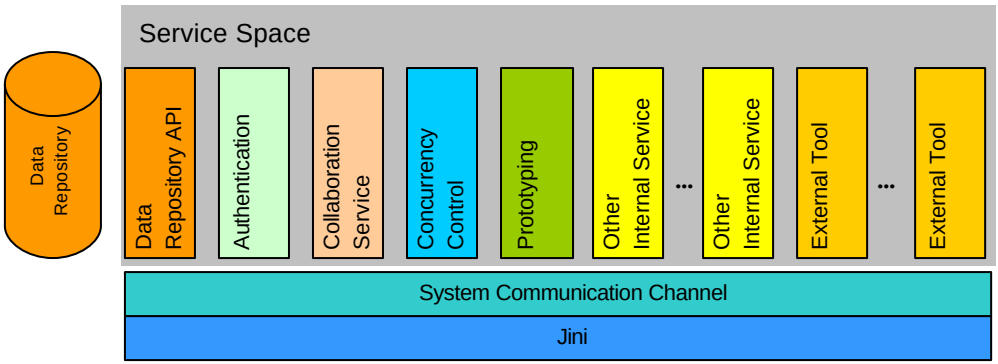

Figura 4. Arquitetura de integração de serviços no Service Space

A integração de serviços no Service Space utiliza estruturas de *proxy*, contribuindo com a transparência na distribuição de recursos no ambiente de rede. Todo cliente utilizando um dado serviço interage apenas com um *proxy* de serviço disponível localmente. O *proxy*, por sua vez, propaga dados e instruções de controle para a verdadeira implementação do serviço sempre que necessário. Essa abordagem é particularmente útil na simplificação da integração de ferramentas externas, bem como na modelagem e execução de *workflows*. Devido à clara separação entre a implementação da ferramenta e sua interface de acesso – definida pelo *proxy* - as ferramentas podem ser consideradas encapsuladas e a integração pode ser feita no nível dos *proxies*.

3 Visualização e Semântica de Projeto

Conforme descrito na seção anterior, tanto o modelo de dados de projeto quanto os módulos de ferramentas de CAD de Cave2 são implementados como *frameworks* orientados a objetos. Esses *frameworks* são inter-relacionados por natureza, pois os módulos de ferramentas de CAD devem poder instanciar, visualizar e modificar os dados de projeto de acordo com a intenção do projetista. Além disso, a presente abordagem explora o inter-relacionamento entre ambos *frameworks* para suportar o acesso colaborativo aos dados de projeto, fazendo com que a semântica do projeto – representada por instâncias de primitivas de dados de projeto – mantenha-se consistente com uma ou mais visualizações do projeto – representadas por módulos de interface gráfica de ferramentas de CAD.

Para permitir múltiplas representações visuais de uma única instância de dados de projeto, o *framework* deve incluir recursos para permitir que os blocos de dados mantenham suas representações visuais coerentes com o seu estado mesmo quando estejam sendo alterados. Por exemplo, se um projetista mantém duas representações visuais de um bloco de projeto – como no caso de duas janelas em um editor de esquemáticos, uma com a visão geral do projeto e outra com os detalhes de um dos blocos – as alterações no modelo de dados resultantes da interação do projetista com uma das representações visuais devem ser notificadas à outra delas para que ela possa atualizar-se e manter-se consistente ao estado do modelo.

Várias das abordagens disponíveis na literatura para manter a consistência entre a informação e suas representações visuais definem a separação explícita entre o modelo da informação e suas visualizações, e prevêem a existência de um mecanismo de consistência que controla as interações entre o modelo, suas visualizações e o mundo externo. Essa arquitetura é a base do *framework* MVC (Model-View-Controller), incluso na linguagem de programação Smalltalk [17] e posteriormente formalizado como o padrão de projeto *Observer* por Gamma et. al. [15].

Desacoplando o modelo e suas visualizações, facilita-se a implementação de várias visualizações diferentes - porém equivalentes – do mesmo bloco de projeto. Para algumas formas particulares de representação onde a visualização incorpora elementos adicionais aos definidos pelo modelo, pode-se ainda flexibilizar o mecanismo de consistência de forma a propagar apenas as alterações que resultem em modificação no estado do modelo. Neste

trabalho, essa possibilidade é explorada na implementação de duas metodologias para colaboração: visualmente acoplada e visualmente desacoplada. Tais metodologias, detalhadas em [21], permitem que múltiplos projetistas compartilhem um modelo de dados de projeto sem que necessariamente compartilhem uma única visualização. Usando a metodologia visualmente desacoplada, pode-se definir quais alterações feitas por um projetista devem ser propagadas aos outros projetistas, permitindo maior flexibilidade no controle de consistência entre visualizações. O único requisito é que as alterações que resultam em mudanças na semântica do modelo de dados de projeto sejam obrigatoriamente propagadas.

No *Framework* Cave2, as primitivas de representação de projeto – portas lógicas e blocos funcionais, por exemplo – são modeladas como instâncias de uma classe concreta, que por sua vez herda parte de sua interface e comportamento de uma classe abstrata. Essa abordagem é comum em *frameworks* orientados a objetos, pois as classes abstratas - mesmo não modelando diretamente nenhum elemento do domínio da aplicação – tem grande importância na organização da hierarquia de classes e na atribuição de comportamento comum a uma classe de objetos. Essa abordagem é usada para incluir em Cave2 o suporte às metodologias colaborativas: as bases do mecanismo de controle de consistência entre o modelo de dados de projeto e suas múltiplas visualizações estão incluídos nas superclasses abstratas do *framework* orientado a objetos – as primitivas de dados de projeto e de elementos de interface gráfica. Assim, todos os modelos de dados projeto usados no Cave2 – incluindo aqueles que serão integrados em atualizações futuras – herdarão tal comportamento.

Aplicando tais conceitos foi possível separar completamente a semântica de projeto de suas formas de visualização, pois são modeladas por objetos diferentes. Assim, é possível permitir múltiplas visualizações – por diferentes projetistas – de um único bloco de dados de projeto. É importante ainda ressaltar a possibilidade de múltiplas formas de visualização – por exemplo um dado bloco de projeto pode ser visto como um esquemático gráfico ou como uma descrição textual na forma de HDL (Hardware Description Language).

Para garantir a consistência entre a semântica do projeto e suas visualizações, assim como entre a semântica de blocos de projeto inter-relacionados, foram usados mecanismos de notificação de atualizações (*update/notify*). Estes mecanismos capturam a interação entre o usuário e uma das visualizações, atualizam a respectiva semântica de projeto e então notificam as demais visualizações para que atualizem-se, a fim de refletir possíveis mudanças.

Um *framework* que implementa a infraestrutura necessária para prover um serviço flexível de notificação foi apresentado por Shen e Sun [18]. Esse *framework* considera um cenário onde vários usuários atualizam simultaneamente um conjunto comum de dados, e as atualizações realizadas por cada usuário devem ser notificadas aos demais. A notificação é dividida em duas partes, a de saída e a de chegada. Para cada usuário, a notificação de chegada representa uma alteração realizada por outro usuário, enquanto a notificação de saída representa a propagação aos outros usuários da alteração realizada localmente. Cada um desses tipos de notificação é caracterizado por sua freqüência e sua granularidade. A freqüência pode assumir uma de três possibilidades – instantânea, escalonada ou definida

pelo usuário – enquanto a granularidade define se a notificação deve ser feita para cada alteração de estado, ou apenas para um sub-conjunto pré-definido de alterações.

Na implementação do sistema aqui descrito, aplicamos um sistema de notificação semelhante ao de Shen e Sun. Inicialmente desenvolvido de forma independente, nossa abordagem foi beneficiada pela visão sistemática do problema apresentada por [18]. O mecanismo resultante foi integrado ao repositório de dados de projeto apresentado a seguir.

4 Repositório de dados de projeto

Um repositório de dados no Cave2 oferece um alto nível de abstração se comparado a bases de dados de projetos inspiradas em consultas ou API (*query-based ou API-based)*. A fim de permitir que os desenvolvedores de CAD se concentrassem somente nos domínios de aplicação da ferramenta de projeto, a abordagem baseada em API foi ampliada com a introdução de uma arquitetura baseada no conceito de persistência transparente [19]. De acordo com esta técnica, o mecanismo de persistência de objetos de dados deve ser escondida sempre que possível. Assim, o cliente pode ter a impressão de estar tratando com objetos regulares em memória, e não com registros em uma base de dados. A arquitetura proposta, incluída no núcleo do *framework* Cave2, permite a gerência direta dos dados dos objetos através de sua própria API - por exemplo, chamando o método *tempblock.addPort(new CaveVisualPort())* ao invés de usar uma API de banco de dados ou uma linguagem de consulta para fazer isso, como o trecho de código abaixo.

```
insert into PORT (portid, name, type) values (64, 'CTRL2', 'in')
insert into PORTBLOCK  (portkey, blockkey) values ('64', '12')
```

O acesso ao repositório é baseado em *proxies*, assim como cada serviço do *Service Space* do Cave2. Para cada ferramenta de projeto que estiver usando o repositório, um *proxy* de serviço é carregado do *Service Space*. Entretanto, tal *proxy* de serviço não é usado diretamente como uma interface completa ao repositório. Seu papel restringe-se à criação, remoção e localização de objetos de projeto. Todas operações restantes são manipuladas pelos *proxies* de objeto, que representam individualmente cada objeto do repositório. A consistência entre o *proxy* de objeto e o respectivo objeto de projeto armazenado no repositório é feita de forma transparente, e está descrita a seguir.

O diagrama de seqüência UML, ilustrado na Figura 5, descreve o procedimento usado no repositório de dados do Cave2. O exemplo descrito supõe que o serviço de repositório já foi encontrado, contatado, e que um *proxy* de serviço já foi carregado. Quando um objeto visual é criado pela ferramenta de projeto, deve ser também criado dentro do repositório. Este papel é desempenhado pelo *proxy* de serviço, que cria a respectiva entrada do objeto de projeto no repositório, criando também um *proxy* local para o objeto de projeto remoto. Cada operação adicional executada através desse objeto visual será delegada ao *proxy* do objeto de projeto, que notificará sua contraparte remota e o seu próprio objeto visual sobre a operação recentemente executada (denotado na Figura 5 como uma chamada ao método *doSomething*).

Do ponto de vista do desenvolvedor da ferramenta, grande parte da funcionalidade desse repositório é transparente. A ferramenta deve obter somente um *proxy* por objeto visual, e lidar com os *proxies* como se estivesse tratando de objetos visuais. Isso simplifica significativamente o desenvolvimento, porque uma ferramenta pode ser desenvolvida primeiramente *standalone*, tratando somente de seus objetos visuais locais, e, adicionalmente, pode ter sua funcionalidade ampliada para utilizar o repositório, mudando somente as chamadas de método dos objetos visuais para os *proxies* do objeto de projeto.

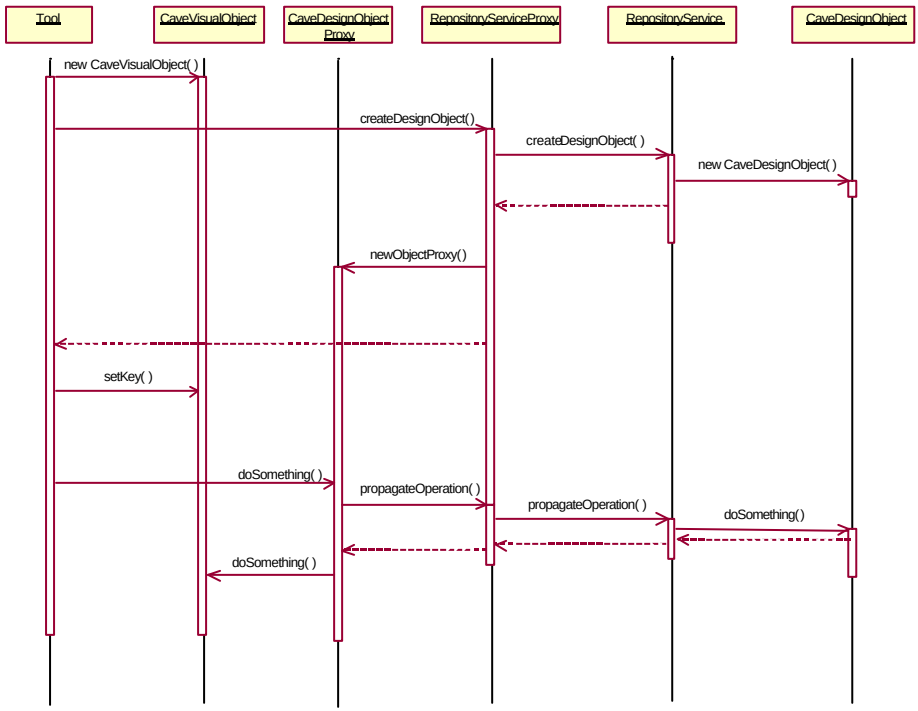

Figura 5. Acesso ao repositório de dados de projeto

A arquitetura Cave2 reduz o problema de manter a consistência dos dados dentro das sessões colaborativas de projeto no que tange a consistência entre a visualização e os objetos do projeto. Em outras palavras, as visões de projeto de cada um dos projetistas devem ser sincronizadas com os dados de projeto no repositório. Como descrito nas seções anteriores, tal consistência é reforçada com a interação entre os objetos visuais e objetos de projeto através dos *proxies*. Nossa abordagem para a sincronização entre os objetos visuais e os objetos de projeto é baseada nas técnicas de atualização/notificação *(update/notify)*, enviadas

através dos objetos *proxies* (Figura 6). Tais atualizações são iniciadas pela interface gráfica da ferramenta de projeto e propagadas para os objetos *proxies* como chamadas de método. Quando seus métodos forem chamados, o objeto *proxy* instancia um objeto evento, o qual encapsula sua chave de identificação, o nome do método chamado e os parâmetros passados. Os parâmetros reais são incluídos somente no objeto de evento quando eles são instâncias de tipos primitivos, tal como *strings* de caracteres ou números. Se um objeto visual for passado como parâmetro, o objeto *proxy* passa sua chave de identificação, pois a serialização do objeto real seria demasiadamente custosa.

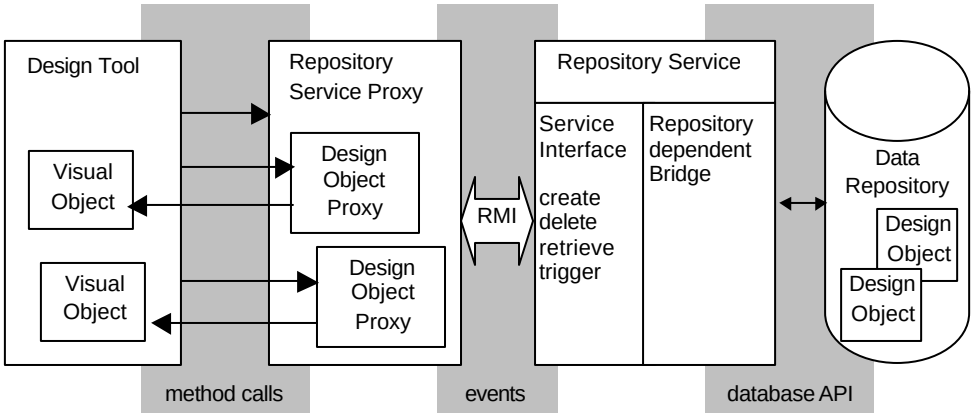

Figura 6. Sincronização entre visualização e semântica de projeto usando *proxies*

O objeto de evento é passado ao *proxy* de serviço, que usa uma conexão RMI para chamar o método que irá disparar o serviço remoto. O objeto de evento é passado como um parâmetro, de tal forma que o repositório pode encontrar o objeto de projeto real dentro de seu índice e chamar o método referenciado através dos parâmetros fornecidos.

A presente implementação dos objetos de evento foi construída através da tecnologia Jini [20]. Ela segue o modelo básico de eventos introduzido na versão 1.1 da linguagem Java. Tal modelo define que um evento consumidor deve se registrar com cada evento produtor com o qual pretenda manter contato. Os eventos produtores devem executar tal procedimento de registro e notificar todos os consumidores caso um de seus estados mude. Um produtor pode produzir os eventos associados a diferentes mudanças de estado. Com isso, os diferentes tipos de eventos podem ser diferenciados pelo atributo *eventID* e pela classe real do objeto do evento. Uma complexidade adicional deve ser manipulada quando os eventos precisam ser enviados através da rede. O modelo de eventos remotos Jini usa Java RMI, que implementa recursos para re-seqüenciamento de eventos e tolerância a falhas na rede. O primeiro é necessário para determinar a ordem correta que os eventos recebidos serão

processados pelo consumidor, enquanto o último - baseado no conceito de exceções - permite a recuperação de eventos perdidos.

Como descrito na Figura 6, a comunicação entre os *proxies* de objeto e suas respectivas contrapartes localizadas no repositório de projeto são baseadas na propagação dos eventos. Entretanto, a implementação de tal comunicação oferece várias alternativas. Os *proxies* podem ser genéricos o bastante para que possam enviar ao repositório todos os eventos recebidos da GUI da ferramenta de projeto, ou ainda encapsular alguma inteligência de forma a verificar a semântica dos eventos, encaminhando somente os válidos ao repositório. A primeira estratégia pode ser utilizada para qualquer tipo de interface com usuário e objeto visual, enquanto a segunda dever ser particularmente implementada a um tipo específico de ferramenta e objeto visual. Em outras palavras, a criação de *proxies* que possam compreender a semântica de eventos envolve a criação de uma instância de um tipo, requerendo a criação prévia - para cada objeto de projeto - de uma classe que defina a interface do *proxy*.

A implementação proposta inclui os *proxies* genéricos, que podem capturar eventos de todas as primitivas de visualização do *framework* Cave2, bem como as suas possíveis extensões. Uma vez capturados, esses eventos podem ser enviados ao repositório remoto de acordo com a disponibilidade do canal de comunicação. Como mencionado anteriormente, tais *proxies* genéricos não executam nenhuma análise semântica dos eventos capturados. Em muitos casos, entretanto, uma análise mais detalhada da semântica é necessária a fim de otimizar a comunicação: os eventos semanticamente inválidos não são propagados ao servidor remoto, e os eventos semanticamente corretos podem ser executados na visualização ao mesmo tempo que estão sendo executados aos objetos remotos do projeto. A fim executar tal análise semântica, são necessários *proxies* de objetos específicos para cada objeto visual. Cada *proxy* de objeto de projeto deve executar os mesmos métodos executados por seu respectivo objeto de projeto, permitindo que o comportamento individual seja executado dentro de cada chamada de método. Na implementação atual, tais classes do *proxy* são codificadas manualmente em um processo tedioso, mas uma automatização adicional pode ser fornecida em um procedimento similar ao processo descrito em [19]. Nesses casos, todos os *proxies* herdam os mecanismos de comunicação implementados pelas classes de *proxy* genérico mencionadas acima, reutilizando assim todos os procedimentos genéricos implementados nas superclasses para a comunicação com o repositório remoto. Somente os métodos específicos à análise semântica devem ser implementados.

A comunicação entre os *proxies* e o repositório pode também ter várias possibilidades de implementação, tais como, conexões dedicadas usando *sockets*, invocação remota de métodos ou eventos distribuídos. Todas as três implementações são suportadas pelo Cave2 e definidas no *Service Space*. O uso de *sockets*, entretanto, requereria a descrição de um protocolo de comunicação completo. O uso de RMI simplifica o desenvolvimento, pois fornece uma interface de alto nível que serve de fundação ao protocolo de comunicação. Entretanto, o uso de uma comunicação RMI entre cada *proxy* de objeto de projeto e suas contrapartes pode ser muito custoso, requerendo uma conexão RMI dedicada por *proxy*, sendo que centenas de *proxies* poderiam estar sendo usados simultaneamente por uma dada ferramenta. Para otimizar tal procedimento, combinamos a abordagem RMI com uma

abordagem de eventos distribuídos. A implementação proposta foi construída com uma única conexão RMI entre o *proxy* de serviço e o repositório. Através desta única conexão RMI, toda a comunicação entre a ferramenta e o repositório foi implementada como uma série de eventos.

5 Estudos de Caso

Três estudos de caso diferentes foram realizados para validar a abordagem proposta, cada um deles envolvendo um sub-conjunto das contribuições do presente trabalho. O primeiro utiliza a arquitetura de distribuição de recursos baseada em proxies para implementar uma plataforma de prototipação usando módulos de hardware reconfigurável. O segundo estende as fundações do framework orientado a objetos visando suportar projeto baseado em interfaces. O terceiro estudo de caso aborda a possibilidade de integração de metadados multimídia ao modelo de dados de projeto. Os dois primeiros estudos de caso serão detalhados nas próximas subseções. O terceiro estudo de caso foi explorado no contexto de uma plataforma online de educação e treinamento descrita em [24].

5.1 Prototipação distribuída usando hardware reconfigurável

Este estudo de caso utiliza o conceito do Service Space para implementar uma plataforma de prototipação usando hardware reconfigurável. Assim como os demais serviços do Service Space, cada módulo de hardware reconfigurável – por exemplo placas de prototipação baseadas em FPGA – é integrado ao mecanismo de *lookup* e é acessível por outras ferramentas utilizando um *proxy*, permitindo que protótipos de sistemas em desenvolvimento sejam validados. A interface de acesso exportada pelo *proxy* permite que um módulo cliente defina a configuração do módulo de hardware e envie dados a ele. Assim, os projetistas de um dado sistema podem utilizar ferramentas de síntese, posicionamento e roteamento para gerar a descrição desse sistema na forma de uma configuração para a plataforma de prototipação. Uma vez configurada através de seu *proxy*, a plataforma de prototipação pode ser submetida a uma série de vetores de teste também enviados através de seu *proxy*.

A Figura 7 mostra com detalhes em um diagrama de sequência UML o procedimento de utilização da plataforma de prototipação. No lado do servidor, a classe *reconfigurable hardware service* representa o serviço de prototipação propriamente dito. Ele utiliza uma intância da classe *reconfigurable hardware backend* para comunicar-se com um tipo específico de plataforma de prototipação (essa classe deve ser estendida usando herança para cada tipo de plataforma de hardware reconfigurável, encapsulando assim suas especificidades). Uma vez que a comunicação com uma determinada plataforma foi estabelecida, o *reconfigurable hardware service* comunica-se com o serviço de *lookup* para registrar-se. O procedimento de registro depende uma chave - enviada por cada um dos serviços disponíveis no servidor de *lookup* - que é capaz de diferenciar as características do serviço registrado. Neste estudo de caso, a chave deve incluir detalhes sobre a plataforma de

prototipação, como sua capacidade lógica, o modelo do seu dispositivo reconfigurável, capacidade de memória, etc.

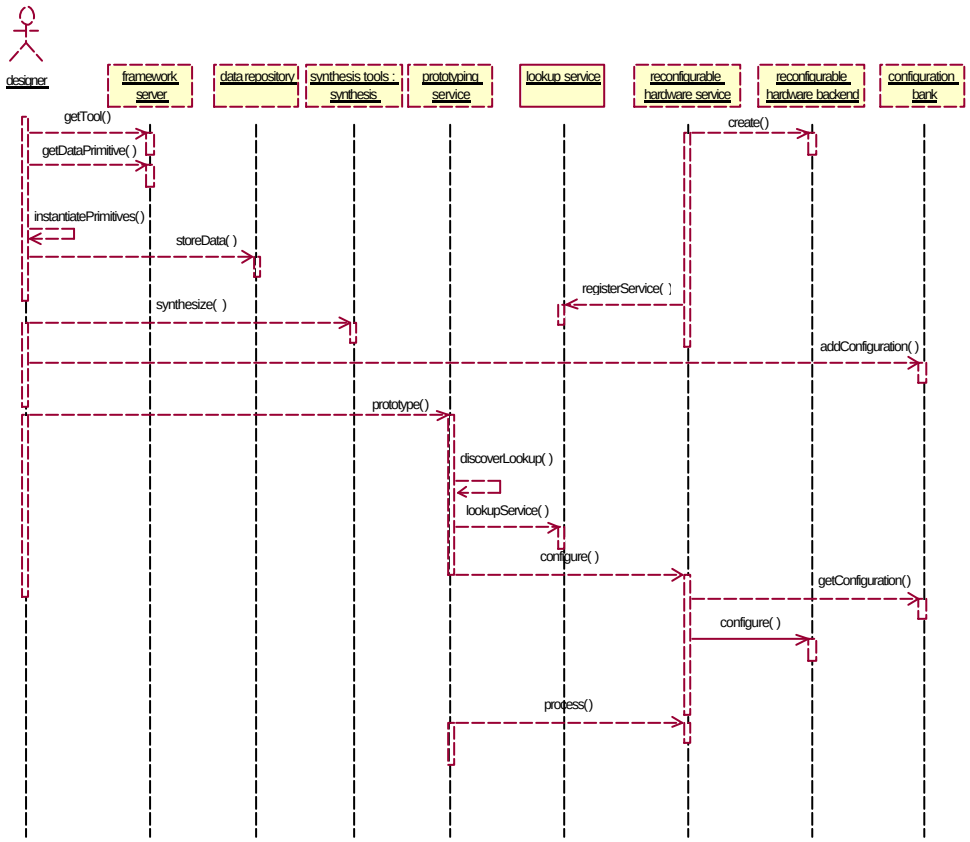

Figura 7. Diagrama de sequência mostrando a utilização da plataforma de prototipação implementada como estudo de caso do Service Space

No lado do cliente, o diagrama mostra o acesso ao servidor do *framework* por parte do projetista, bem como o carregamento de primitivas de dados de projeto e módulos de ferramentas (o mesmo procedimento também descrito na Figura 1). Uma vez que o sistema sendo projetado é mapeado em uma configuração para a plataforma de prototipação, essa configuração é armazenada numa instância da classe *configuration bank*.

O acesso por parte do cliente à plataforma de prototipação é feito através do Service Space. Primeiramente, o cliente busca um servidor de *lookup* (pode haver mais de um *lookup* em um Service Space) e nele faz a busca de uma plataforma de hardware configurável para prototipar o sistema sendo projetado. Essa busca é feita usando os parâmetros presentes na chave usada pelas instâncias de *reconfigurable hardware service* durante o procedimento de registro. Dessa forma, o projetista pode buscar todas as plataformas de prototipação disponíveis no Service Space que satisfaçam seus critérios e então escolher aquela com melhor usabilidade (por exemplo aquela com menor latência de acesso na rede, ou aquela que possibilite um feedback visual com fotos ou stream de video). Uma vez escolhida a plataforma, o projetista recebe do Service Space um *proxy* do respectivo serviço e através dele configura a plataforma passando uma referência à configuração desejada (armazenada previamente no *configuration bank*). Também através do proxy é conduzido o teste ao sistema prototipado, através do envio dos vetores de teste a serem carregados na memória da plataforma de prototipação. Uma vez processados, os resultados do teste são lidos da memória e enviados ao cliente para análise.

É importante notar que no diagrama da Figura 7 o objeto proxy não aparece explicitamente, pois ele não contribui com a funcionalidade da plataforma de prototipação, e sim provê a abstração da distribuição dos seus recursos. Já a Figura 8 mostra explicitamente o funcionamento do *proxy*. A Figura 9 mostra uma das plataformas de prototipação implementadas. Mais detalhes sobre este estudo de caso podem ser encontrados em [22].

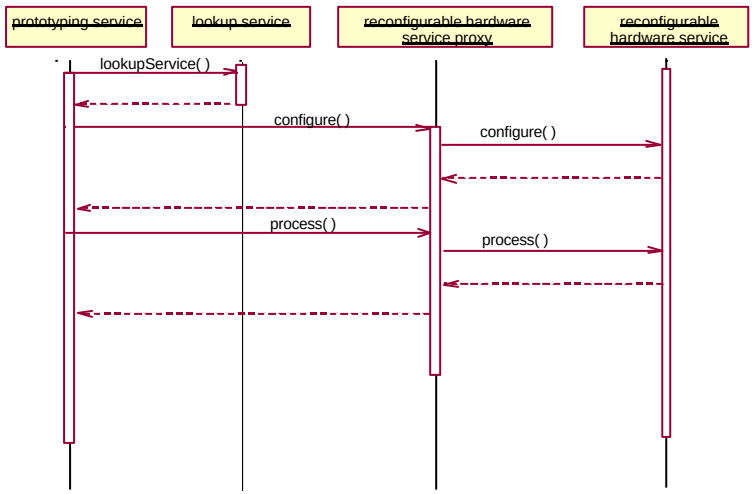

Figura 8. Diagrama de sequência mostrando o acesso às plataformas de prototipação usando *proxies*

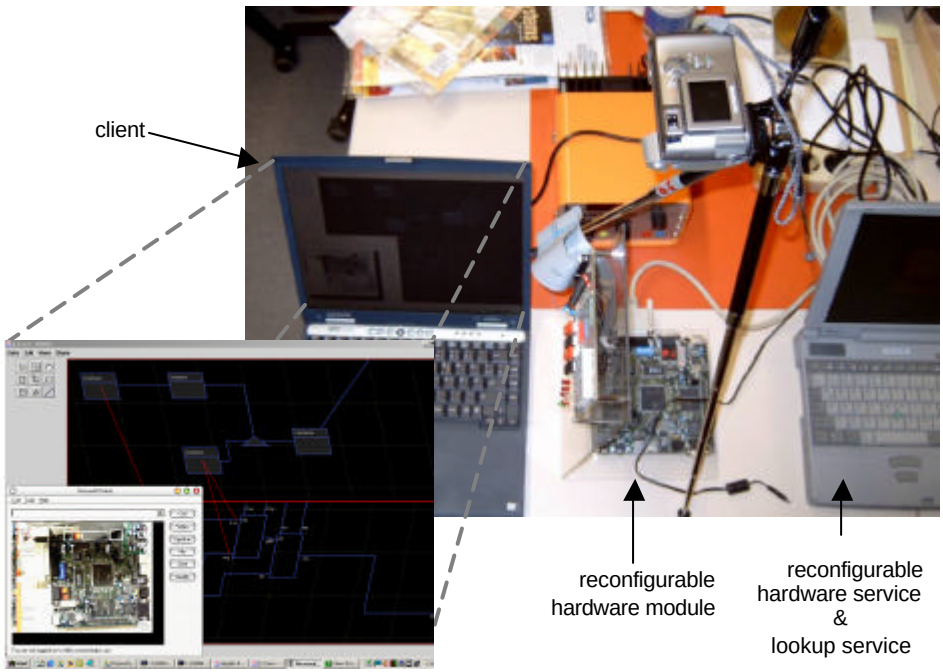

Figura 9. Implementação de uma plataforma de prototipação

5.2 Suporte ao profeto basendo em interfaces

Este estrudo de caso estende as fundações do framework orientado a objetos descrito na subseção 3.1 visando suportar projeto baseado em interfaces (Interface-based design [16]). Essas extensões - primitivas de representação de projeto e partes de ferramentas - são usadas na implementação de uma ferramenta chamada IBlaDe, que permite a criação colaborativa de modelos funcionais e estruturais de sistemas integrados.

A Figura 10 mostra as extensões ao framework implementado originalmente (mostrado na Figura 3), visando suportar os conceitos de modelagem baseada em interfaces. Nota-se que no modelo extendido de dados de projeto é possível definir interfaces - tanto providas quanto requeridas - para cada porta. A adição da classe *InternalPort* permitiu a composição hierárquica da definição de interface seguendo o mesmo padrão *Composite* presente no framework original. Essa solução permitiu que a interface de um dado bloco seja estaticamente definida pelas interfaces dos blocos que o compõe. O tipo de cada bloco de projeto também passou a ser definido mais concretamente através do conjunto de assinaturas dos processos que ele deve ser capaz de executar.

O modelo estendido de dados de projeto foi validado através de uma ferramenta de modelagem chamada IBlaDe. Essa ferramenta foi construída a partir dos módulos disponíveis no framework orientado a objetos descrito na subseção 3.1, somados a um recurso para a melhor visualização dos dados de projeto baseado em interfaces: um editor simultâneo de diagramas UML e esquemáticos de blocos funcionais. A Figura 11 mostra a interface gráfica dessa ferramenta.

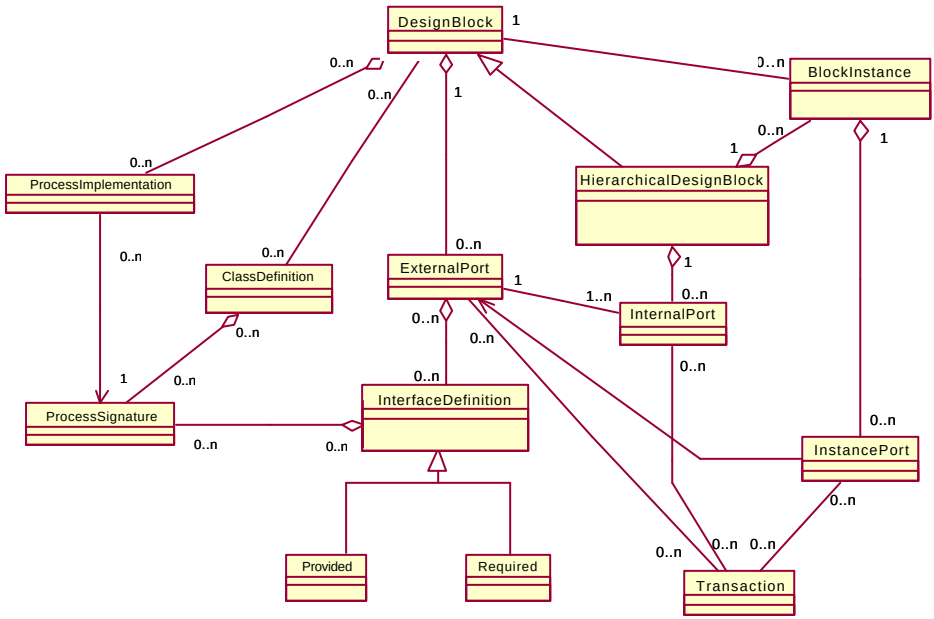

Figura 10. Extensão do framework de primitivas de dados de projeto visando suportar projeto baseado em interfaces

Este estudo de caso mostra o potencial de extensão e atualização do *framework* desenvolvido no presente trabalho. É importante notar que como as primitivas de dados de projeto desenvolvidas na extensão do *framework* herdam a funcionalidade definida nas classes fundamentais, todos os recursos de sincronização com múltiplas visualização, consistência nos repositórios de dados e demais recursos de suporte à colaboração também são herdados, mostrando assim que o desenvolvimento de ferramentas para projeto colaborativo é significantemente simplificado com o uso da abordagem apresentada no presente trabalho. Mais detalhes sobre este estudo de caso podem ser encontrados em [23].

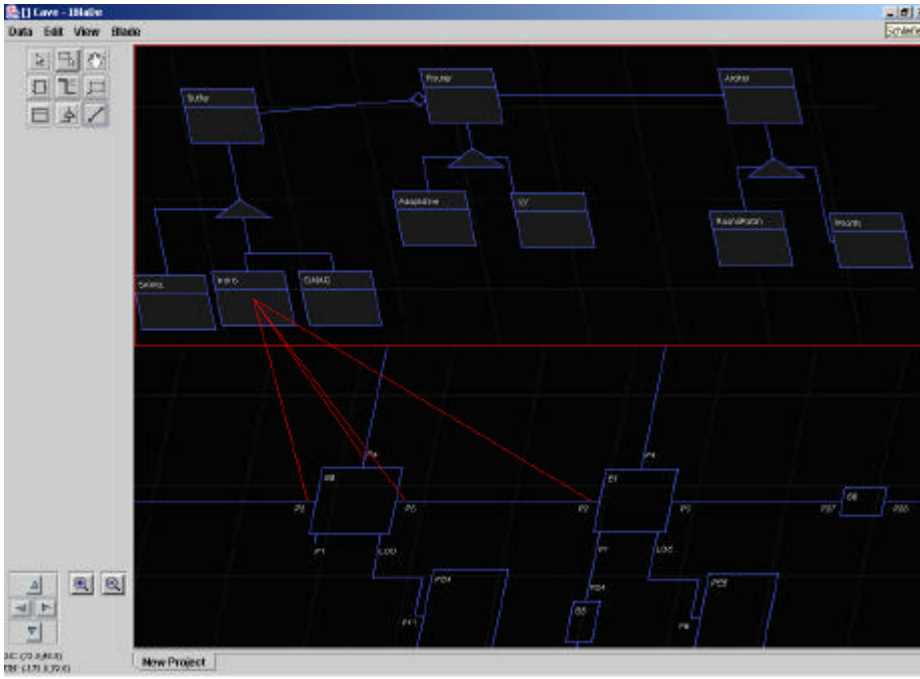

Figura 11. Interface gráfica de IBlaDe

6 Considerações Finais

O CAD Framework implementado neste trabalho foi chamado Cave2 e seguiu a clássica arquitetura em níveis apresentada por Barnes, Harrison, Newton e Spickelmier [12]. Durante o projeto e a implementação do Cave2, uma série de avanços em relação às abordagens anteriores foi obtida com a exploração das vantagens advindas do uso de um framework orientado a objetos:

- uma vez que frameworks orientados a objetos são extensíveis por definição, o mesmo pode ser dito a respeito da implementação no Cave2 dos conjuntos de primitivas de dados de projeto, bem como de blocos para a construção de ferramentas de CAD. Isso implica que tanto o modelo de representação de projeto quanto os módulos de software processando tal modelo podem ser atualizados ou adaptados para uma metodologia de projeto específica, e que essas atualizações e adaptações ainda herdarão os aspectos arquiteturais e funcionais implementados nos elementos básicos do framework orientado a objetos;

- ambos os aspectos relativos à semântica do projeto e à visualização do projeto são partes do framework orientado a objetos, mas em modelos claramente separados. Isso possibilita o uso de várias estratégias para a visualização de um conjunto de dados de projeto, o que dá aos participantes de uma sessão de projeto colaborativo a flexibilidade de escolha individual de estratégia de visualização;

- o controle de consistência entre semântica e visualização - uma questão particularmente importante em um ambiente de projeto onde coexistem múltiplas visualizações de cada projeto - também está incluído nas fundações do framework orientado a objetos implementado. Esse mecanismo é genérico o bastante para ser usado também pelas possíveis extensões do modelo de dados de projeto, uma vez que ele é baseado na inversão de controle entre a visualização e a semântica. A visualização recebe a intenção do usuário e propaga esse evento ao modelo da semântica, o qual avalia a possibilidade de uma mudança de estado. Se positivo, ele dispara a mudança de estado em ambos os modelos de visualização e semântica. A abordagem proposta nesta tese usa tal inversão de controle para incluir um nível adicional de processamento entre a semântica e a visualização, visando o controle de consistência nos casos de múltiplas visualizações;

- para otimizar o mecanismo de controle de consistência entre semântica e visualização, uma abordagem baseada em eventos foi proposta, buscando discretizar cada interação entre o projetista e suas visualizações do projeto. A informação sobre cada uma das interações é encapsulada em um objeto-evento, que pode ser propagado para o modelo da semântica do projeto - e então para as demais possíveis visualizações - de acordo com a política de consistência que esteja sendo usada. Além disso, o uso de eventos permite que as interações do usuário com a visualização sejam acumuladas para uma posterior sincronização com a semântica do projeto, caso haja indisponibilidade de conexão entre elas;

- o uso de objetos de *proxy* aumentou significativamente o nível de abstração da integração de recursos de automação de projeto, pois tanto ferramentas e serviços remotos quanto os instalados localmente são acessados através de chamadas de métodos em um objeto local. A conexão aos serviços e ferramentas remotos é obtida através de um protocolo de look-up, abstraíndo completamente a localização de tais recursos na rede e permitindo a adição e remoção em tempo de execução;

- o CAD Framework foi implementato completamente usando a tecnologia Java, usando dessa forma a Java Virtual Machine como intermediário entre o sistema operacional e o CAD Framework, garantindo dessa forma a independência de plataforma. As limitações impostas pelo uso de Java - por exemplo a impossibilidade de definição de herança múltipla entre classes e a redução de desempenho devido à interpretação do código em formato intermediário - não tiveram impacto significativo no desenvolvimento do presente trabalho. A redução de desempenho seria significativa se houvesse a necessidade da implementação de todas as ferramentas de apoio ao projeto usando Java. Na abordagem proposta, as ferramentas de execução computacionalmente intensiva bem como repositórios para grandes quantidades de dados são encapsulados e integrados ao Service Space, mantendo assim sua performance original. Já a ausência de herança múltipla foi compensada com o uso de

interfaces, um recurso da linguagem Java que permite a um objeto assumir múltiplos tipos externos a sua hierarquia de herança, bem como com padrões de projeto tais como *Bridge* e *Chain of Responsibility* [15].

Todas as contribuições listadas anteriormente contribuiram com o aumento do nível de abstração da distribuição de recursos de automação de projeto e também apresentaram um novo paradigma para a interação remota entre projetistas. O CAD Framework no qual tais contribuições foram aplicadas é capaz de suportar colaboração de granularidade fina baseada em eventos, onde cada atualização feita por um projetista pode ser propagada para o restante da equipe, mesmo que estejam todos geograficamente distribuídos. Isto pode aumentar a sinergia de grupo entre os projetistas e permitir uma troca mais rica de experiências entre eles, aumentando significativamente o potencial de colaboração quando comparado com abordages baseadas em acesso a arquivos e registros propostas anteriormente.

A Tabela 2 compara os avanços obtidos no presente trabalho com abordagens afins encontradas na literatura. As colunas marcadas com "S" representam a possibilidade de cada um dos ambientes de suportar completamente a característica listada. As colunas marcadas com "E" mostram que a o ambiente é capaz de inter-operar com ferramentas externas para implementar a característica listada.

Tabela 2. Comparação entre ambientes multi-usuário de projeto automatizado de sistemas integrados que suportam distribuição de recursos em rede

Ferramenta	Suporte a gerência de metodologias usando *workflow*	Suporte a versões	Abstrai a complexidade da distribuição em rede dos recursos de CAD	Independente de plataforma	Suporta extensibilidade do modelo de dados de projeto
Nelsis [13]	S	S			
Version Server [6]		S			
STAR [8]		S			
Ulysses & Odyssey [25]	S	S			
WELD [26]	S		S	S	
OmniFlow [9]	S		S	S	
Moscito [10]	S		S	S	
PPP [27]			S	S	
JavaCAD [28]			S	S	
Ptolemy II [29]	E			S	S
Cave [11]			S	S	
Cave2	E	S	S	S	S

Referências

[1] SEMICONDUCTOR INDUSTRY ASSOCIATION. International Technology Roadmap for Semiconductors: 1999 edition. Austin: International SEMATECH, 1999.

[2] BOREL, J. et al. The MEDEA+ Design Automation Roadmap. Paris: MEDEA+ Office, 2002.

[3] BROWN, S. Law of accelerating returns. Midyear Forecast, EE Times Special Report, 2000.

[4] INDRUSIAK, L. S. A Review on the Framework Technology Supporting Collaborative Design of Integrated Systems. Exame de Qualificação. Porto Alegre: PPGC UFRGS, 2002. 108 p.

[5] JOHANSEN, R. "Groupware: Computer support for business teams". New York: The Free Press, 1988.

[6] KATZ, R. H. Towards a unified framework for version modeling in engineering databases. In: ACM Computing Surveys. Vol. 22, No. 4, December 1990. p. 375-408.

[7] HARRISON, D. S. et al. Data management and graphics editing in the Berkeley Design Environment. In: Proceedings of the IEEE International Conference in Caputer Aided Design, 1986.

[8] WAGNER, F. R.; LIMA, A.H.V. Design Version Management in the GARDEN Framework. In : Proceedings of the 28th Design Automation Conference, ACM/IEEE, June 1991. p. 704-710.

[9] BRGLEZ, F.; LAVANA, H. A Universal Client for Distributed Networked Design and Computing. In: Proceedings of the 38th Design Automation Conference, 2001. Los Alamitos: IEEE Computer Society, 2001.

[10] SCHNEIDER, A. et al. Internet-Based Collaborative Test Generation with MOSCITO. In: Proceedings of Design, Automation and Test in Europe, Paris, 2002. p. 221- 226.

[11] INDRUSIAK, L. S.; REIS, R. A. L. From a Hyperdocument-Centric to an Object-Oriented Approach for the Cave Project In: XIII Symposium on Integrated Circuits and System Design, 2000, Manaus. Proceedings. Los Alamitos: IEEE Computer Society, 2000. p.125 – 130.

[12] BARNES, T. J.; HARRISON, D.; NEWTON, A. R.; SPICKELMIER, R. L. Electronic CAD Frameworks. Boston: Kluwer Academic Publishers, 1992. 196 p.

[13] VAN DER WOLF, P.; BINGLEY, P.; DEWILDE, P. On the Architecture of a CAD Framework: The NELSIS Approach. In: Proceedings of the European Design Automation Conference, 1990. p. 29-33.

[14] JOHNSON, R.; FOOTE, B. Designing Reusable Classes. Journal of Object-Oriented Programming, Vol 1 (2), 1988, pp. 22-35.

[15] GAMMA, E. et al. "Design Patterns: elements of reusable object-oriented software". Reading: Addison Wesley, 1995.

[16] INDRUSIAK, L. S., REIS, R. A. L., GLESNER, M. Supporting Consistency Control between Functional and Structural Views in Interface-based Design Models In: Proceedings of the Forum on Design Languages, 2003, Frankfurt.

[17] KRASNER, G. E.; POPE, S.T. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, 1(3):26–49, Aug./Sep. 1988.

[18] SHEN, H.; SUN, C. Flexible notification for collaborative systems. In: Proc. of ACM Conference on Computer Supported Cooperative Work, 2002. p. 77-86.

[19] ROOS, R. M. "Java Data Objects". London: Addison-Wesley, 2003. 264 p.

[20] LI, S. "Professional Jini". Birmingham: Wrox Press, 2000. 886 p.

[21] INDRUSIAK, L. S.; BECKER, J.; GLESNER, M.; REIS, R. A. L. Distributed Collaborative Design over Cave2 Framework. In: 11th International Conference on Very Large Scale Integration of Systems-on-Chip, 2002, Montpellier. SOC Design Methodologies. Boston : Kluwer Academic Publishers, 2001. p. 97-108.

[22] INDRUSIAK, L. S.; LUBITZ, F.; GLESNER, M.; REIS, R. A. L. Ubiquitous Access to Reconfigurable Hardware: Application Scenarios And Implementation Issues. In: Design Automation and Test in Europe (DATE), 2003, Munich. Proceedings. Los Alamitos: IEEE Computer Society, 2003. p.940 – 945.

[23] INDRUSIAK, L. S.; REIS, R. A. L.; GLESNER, M. Supporting Consistency Control between Functional and Structural Views in Interface-based Design Models. In: Forum on Design Languages, 2003, Frankfurt. Proceedings. Gières: ECSI, 2003. CDROM.

[24] INDRUSIAK, L. S.; GLESNER, M.; REIS, R. A. L.; ALCÁNTARA, G. P.; HOERMANN, S.; STEINMETZ, R. Reducing Authoring Costs of Online Training in Microelectronics Design by Reusing Design Documentation Content. In: International Conference on Microelectronic Systems Education, 2003, Anaheim. Proceedings. Los Alamitos: IEEE Computer Society, 2003. p.57 – 58.

[25] BROCKMAN, J.B.; COBOURN, T.F.; JACOME, M.F.; DIRECTOR, S.W. The Odyssey CAD Framework. IEEE DATC Newsletter on Design Automation, Spring 1992.

[26] CHAN, F.; SPILLER, M.; NEWTON, R. WELD - An Environment for Web-Based Electronic Design. In: DESIGN AUTOMATION CONFERENCE, 1998. Proceedings. Los Alamitos: IEEE Computer Society, 1998. p. 146-152.

[27] BENINI, L.; BOGLIOLO, A.; DE MICHELI, G. Distributed EDA tool integration: the PPP paradigm. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN. Proceedings. [S.l.: s.n.], 1996. p. 448-453.

[28] DALPASSO, M.; BOGLIOLO, A.; BENINI, L. Specification and validation of distributed IP-based designs with JavaCAD. In: Design Automation and Test in Europe (DATE), 1999, Munich. Proceedings. Los Alamitos: IEEE Computer Society, 1999. p.684 – 688.

[29] LEE, E. A. et al. Heterogeneous Concurrent Modeling and Design in Java. Memorandum UCB/ERL M04/17. Berkeley: EECS, University of California, 2004.