OpenBus – Uma Arquitetura para Integração de Aplicações

1 Introdução

Os profissionais da área de Exploração e Produção da Petrobras utilizam diversas aplicações computacionais no desempenho de suas atividades técnicas. Por exemplo, as tarefas relacionadas à aquisição, processamento e interpretação de dados sísmicos exigem a participação de diferentes profissionais, e cada profissional pode necessitar de várias aplicações. O fato de uma única aplicação não dar suporte a todas as fases do trabalho e, às vezes, nem mesmo a uma fase inteira, é fruto da elevada complexidade técnica inerente à manipulação dos dados. Essa complexidade se reflete diretamente nas aplicações, forçando um alto grau de especialização, e praticamente torna impossível a construção de uma aplicação que permeie todas as fases de trabalho sobre um dado.

Uma decorrência direta dessa pluralidade de aplicações utilizadas para a manipulação de um dado é a necessidade de integração. Afinal, para que o trabalho seja realizado, é preciso que os dados sejam trabalhados em uma aplicação, migrados para outra aplicação, retrabalhados e assim sucessivamente até que o resultado final seja alcançado. Em resumo, a capacidade de integração entre aplicações é um suporte essencial para o trabalho dos profissionais da área de E&P.

Ao longo de seu fluxo de trabalho, um profissional carrega um dado em uma aplicação e, através das funcionalidades providas por esta aplicação, manipula o dado como necessário. Em uma situação típica, o profissional manipula o dado até o limite oferecido pela aplicação e, então, transporta o dado para outra aplicação de forma a continuar seu trabalho. Entretanto, na prática, essa troca de dados entre duas aplicações não necessariamente é feita de forma simples ou até mesmo satisfatória.

Diferentes aplicações da área de E&P possuem formas próprias de representar e armazenar os dados. Além disso, a quantidade e a variedade de atributos e meta-informações relacionadas a um mesmo tipo de dado também podem ser diferentes. A falta de padronização gera dificuldades na troca de dados entre as aplicações que, muitas vezes, acaba sendo feita manualmente pelo próprio usuário, externamente às aplicações envolvidas. A integração externa, ou manual, consiste na exportação dos dados em algum formato padronizado de arquivo na aplicação de origem, a carga desse arquivo na aplicação destino e os ajustes necessários para que eventuais informações adicionais, não representadas no arquivo gerado, sejam criadas corretamente no destino. Essa forma de integração é fraca, pois obriga o usuário a assumir o papel de integrador e abre a possibilidade de perda das informações que porventura não sejam representadas no formato de arquivo utilizado. Em uma situação extrema, as aplicações podem não compartilhar um formato comum de arquivo, o que pode inviabilizar a troca de dados entre elas.

Quando as aplicações são desenvolvidas internamente pela Petrobras, ou possuem código aberto, ou, ainda, dispõem de interfaces para programação de extensões, a integração pode ser feita de forma programática direta. Essa integração *ad hoc* entre duas aplicações facilita substancialmente o trabalho do usuário que, ao invés de trocar arquivos e configurações entre as aplicações, apenas comanda a transferência dos dados. Além disso, a integração programática pode considerar, além dos dados, os atributos e meta-informações que sejam comuns, ou convertíveis, entre as aplicações, maximizando a qualidade do dado trocado e diminuindo as eventuais perdas de informações desse processo. Essa forma de integração também está presente, eventualmente, entre aplicações de um mesmo fabricante, como é o caso das aplicações de geologia e geofísica da Landmark.

Dois exemplos de integração programática direta entre aplicações próprias da Petrobras são a integração v3o2/WebSintesi e a WebSintesi/SIGEO. A integração v3o2/WebSintesi é feita através de uma interface de acesso à área de projetos própria do WebSintesi que é acessada pelo v3o2. Dessa forma, o v3o2 tem acesso aos projetos do WebSintesi, podendo ler e escrever arquivos nos projetos. A integração WebSintesi/SIGEO é feita de forma análoga: uma interface de acesso específica para os projetos do SIGEO é acessada pelo WebSintesi que, através dela, consegue criar novos objetos nos projetos SIGEO.

A integração direta entre aplicações é, sem dúvida, melhor do que a integração externa, feita pelo usuário através da troca de arquivos e ajustes manuais. Entretanto, o esforço necessário na infra-estrutura para integrar as aplicações é muito grande, pois precisamos codificar uma *ponte* específica para ligar cada par de aplicações. Isso nos leva a um número quadrático de codificações de pontes em relação ao número de apli-

cações. A cada nova aplicação a ser integrada, é preciso codificar uma ponte que integre esta nova aplicação com cada uma das aplicações já existentes, o que significa ($\frac{n(n-1)}{2}$) codificações para n aplicações. Por exemplo, para integrar 10 aplicações precisamos codificar 45 pontes.

Se a integração direta entre os pares de aplicações demanda um esforço muito elevado, a adoção de aplicações de um mesmo fabricante, que possuem funcionalidades de integração intrínsecas, é direta mas insuficiente. Afinal, nenhum fabricante possui um conjunto de aplicações que atenda a totalidade das funcionalidades exigidas pelos profissionais nos seus fluxos de trabalho. Por outro lado, mesmo que um fabricante possuísse tal conjunto de aplicações, dificilmente sua adoção seria desejável, dada a total dependência que se estabeleceria deste fabricante.

Uma abordagem alternativa para as integrações externa (via arquivos) e direta (via pontes entre os pares) é a adoção de uma plataforma comum de integração entre aplicações. O princípio básico de uma plataforma de integração é definir um componente ou um *barramento* comum através do qual todas as aplicações possam trocar dados. Nesse modelo, cada aplicação se conecta ao barramento, segundo um protocolo próprio deste barramento, e carrega ou exporta dados para outras aplicações também conectadas ao barramento.

Como na integração direta, é necessário codificar uma ponte para conectar cada aplicação ao barramento. Também como na integração direta, o fato de se estabelecer uma interface e de se codificar uma ponte específica, permite maximizar a qualidade do dado trocado entre as aplicações, minimizando a eventual perda de informações decorrente da heterogeneidade dessas aplicações. Por outro lado, como o barramento define uma linguagem comum entre todas as aplicações, a codificação de uma ponte entre uma aplicação e o barramento é o suficiente para que esta aplicação se integre a todas as demais aplicações conectadas ao barramento. Ou seja, diferente da abordagem de integração direta, onde o esforço de infra-estrutura é quadrático, a integração por meio de um barramento exige um esforço linear em função do número de aplicações. De fato, para integrar n aplicações, é preciso codificar apenas n pontes.

Outra vantagem significativa da integração através de um barramento comum é a possibilidade de criarmos *ligações ativas* entre as aplicações. Uma ligação ativa é uma ligação através da qual uma aplicação pode perceber mudanças nos dados de outra aplicação. Ou seja, por meio de uma ligação ativa duas aplicações podem trocar mensagens ao invés de apenas dados. Através das ligações ativas podemos criar integrações onde as aplicações trocam informações dinamicamente, à medida em que o usuário interage com cada uma. Por exemplo, se uma aplicação permite a visualização de um dado proveniente de outra aplicação, é possível refletir na tela uma modificação

que seja feita no dado original imediatamente após essa modificação ter sido feita na aplicação que hospeda esse dado. Além disso, podemos explorar essa forma de comunicação entre as aplicações para, por exemplo, permitir o trabalho colaborativo entre usuários.

Na área de E&P, existem soluções de mercado que implementam arquiteturas de integração de aplicações. Entretanto, como veremos na seção seguinte, essas soluções são limitadas e não atendem a todos os requisitos de integração do E&P da Petrobras. Em função dessas limitações, e do fato desta abordagem de integração se mostrar como a mais adequada, nós propomos, neste documento, a construção de uma arquitetura de integração baseada em um barramento, que chamamos de OpenBus.

Na seção seguinte, além da apresentação e da análise das soluções de mercado, nós apresentamos as tecnologias que podem ser utilizadas na construção de uma arquitetura de integração e avaliamos cada uma em função dos requisitos impostos pelas aplicações de E&P. Na Seção 3, nós justificamos a arquitetura OpenBus, explicitando os requisitos contemplados. Na Seção 4, apresentamos um detalhamento dos serviços que serão providos pelo OpenBus e, por fim, na Seção 5, concluímos este documento, sintetizando os benefícios da arquitetura proposta.

2 Estado da arte

2.1 Soluções de mercado para a integração de dados e aplicações de E&P

A solução para a integração de aplicações de E&P mais abrangente disponível no mercado é a plataforma OpenSpirit [5]. O desenvolvimento dessa plataforma resultou de um consórcio — a OpenSpirit Alliance — que uniu esforços de diferentes companhias de petróleo e fabricantes de software para a definição de um *framework* de integração de aplicações que suportasse tanto o compartilhamento de dados mantidos em diferentes repositórios, como também a colaboração entre diferentes aplicações. Esse *framework* é comercializado hoje por uma companhia de software independente — a OpenSpirit Corporation — que conta com um grupo de investidores que inclui tanto companhias de petróleo, como Chevron e Shell, como também provedores de software e serviços, como Schlumberger e Paradigm.

O *framework* OpenSpirit é composto por três elementos principais: o *OpenSpirit*

Runtime, conectores de dados (*Data Connectors*) e adaptadores de aplicações (*Application Adapters*).

- OpenSpirit Runtime

  O OpenSpirit Runtime provê a infraestrutura de suporte para a integração de aplicações e dados. Essa infraestrutura oferece uma série de serviços utilizados pelos conectores de dados e adaptadores de aplicações, como, por exemplo, a conexão de uma aplicação ao ambiente de integração, a disponibilização de informações sobre os repositórios de dados acessíveis, e a difusão de mensagens (eventos de colaboração) entre aplicações.

  Para garantir a independência da plataforma de execução e da linguagem de desenvolvimento das aplicações — um requisito essencial à ampla interoperabilidade pretendida pelo *framework* — a infraestrutura de suporte de integração implementada pelo OpenSpirit é baseada no *middleware* CORBA, e em alguns dos serviços padronizados para sua arquitetura.

- Conectores de Dados

  Conectores de dados, ou *data servers*, são os elementos responsáveis por implementar os serviços de acesso aos diferentes repositórios, ou *datastores*. Esses conectores provêem às aplicações uma visão uniforme e homogênea dos dados acessados, independente dos formatos internos adotados por cada tipo de repositório. Essa visão uniforme é baseada em um modelo de dados genérico (*OpenSpirit Data Model*), oferecido pelos conectores de dados em sua interface de acesso; é responsabilidade de cada tipo de conector realizar o mapeamento desse modelo para um repositório específico.

  É importante observar que o modelo de dados atualmente implementado pela OpenSpirit é restrito ao domínio da Geologia e Geofísica, o que impede a integração de aplicações de outros domínios, como Engenharia e Química. Além disso, apenas a própria OpenSpirit desenvolve e comercializa conectores de dados; não existe um *toolkit* para o desenvolvimento desses conectores. Essa restrição impede que repositórios de dados providos por aplicações e sistemas desenvolvidos pela Petrobras sejam incorporados ao ambiente de integração, a menos que a própria OpenSpirit seja contratada para desenvolver os conectores necessários.

  A OpenSpirit disponibiliza, atualmente, conectores para os seguintes repositórios: OpenWorks/Seisworks, GeoFrame/IESX/Charisma, GOCAD, Kingdom, Petra, Finder, Managed SEGY (dados SEGY 2D e 3D armazenados em um sistema de arquivos), Recall e ArcSDE. Contudo, conectores específicos podem

não suportar todos os tipos de dados de um repositório; o conector para o GOCAD, por exemplo, oferece por enquanto apenas o acesso a volumes sísmicos.

- Adaptadores de Aplicações

  Adaptadores de aplicações (ou *application plug-ins*), são os componentes responsáveis por integrar as diferentes aplicações ao ambiente OpenSpirit. Esses componentes utilizam a API oferecida pelo *toolkit* de desenvolvimento para prover às aplicações o acesso aos serviços de dados e de colaboração disponibilizados pelo *framework*. O *toolkit* de desenvolvimento está disponível atualmente para diferentes linguagens (Java, C++, C#, VB, Delphi) e plataformas (Windows, Solaris, Linux, Irix). Diversos fornecedores de software, como Schlumberger (Petrel), Landmark (OpenWire), Earth Decision (Gocad) e Hampson-Russell (AVO), provêem adaptadores para suas aplicações. A própria Petrobras é licenciada como desenvolvedor OpenSpirit, e está implementando para o Web-Sintesi/VGE o acesso, via *framework* OpenSpirit, a repositórios de dados como OpenWorks/Seisworks e GeoFrame/IESX.

  Como vimos anteriormente, os serviços de acesso a dados são oferecidos por conectores específicos para cada repositório. Através de um mecanismo de *query* oferecido pela API, aplicações podem criar, ler, atualizar e remover dados com base no modelo de dados genérico oferecido por esses conectores. O mecanismo de *query*, contudo, não permite o acesso a *bulk data*, como volumes sísmicos e horizontes 3D. Para esse tipo de acesso, a API provê a representação dos dados como objetos remotos.

  A colaboração entre aplicações é baseada em um serviço de difusão de mensagens (notificações), oferecido pelo OpenSpirit Runtime. Esse serviço permite que aplicações que operam sobre um mesmo conjunto de dados compartilhem *eventos* de interação de usuário (seleção e alteração de dados, movimentação de cursor, definição de áreas de interesse). A API provida pelo *toolkit* oferece mecanismos para que as aplicações possam gerar e receber esses eventos, encapsulando o acesso ao serviço de notificações.

  Um fator limitante para a colaboração entre aplicações através do *framework* é a restrição do escopo dessa colaboração a uma sessão OpenSpirit. Essa sessão, determinada no momento de conexão da aplicação ao ambiente, estabelece também o conjunto de dados, ou projetos (*ProjectSet*) acessíveis pela aplicação durante a conexão. No ambiente de integração OpenSpirit, cada usuário (*userid*) é associado a uma configuração particular, que inclui a definição das sessões às quais as aplicações iniciadas por esse usuário poderão conectar-se. Como sessões não são compartilhadas entre usuários, não há colaboração entre aplicações iniciadas por usuários diferentes.

Além de eventos de colaboração, o *framework* OpenSpirit permite que aplicações recebam notificações de eventos de alteração dos dados de um repositório (criação, alteração e remoção), enviada pelo conector correspondente. Essa facilidade depende da capacidade do conector gerar esse tipo de evento; atualmente, apenas os conectores OpenWorks e Geoframe possuem essa capacidade.

Uma outra solução disponível no mercado para a integração de aplicações e repositórios de dados de E&P é o PowerHub, comercializado pela Landmark. Assim como o modelo de dados genérico implementado pelo *framework* OpenSpirit, a plataforma PowerHub oferece um modelo de dados que abstrai formas de armazenamento proprietárias, permitindo às aplicações acessar de maneira uniforme dados providos por diferentes repositórios.

Diferentemente do *framework* OpenSpirit, é possível desenvolver conectores para a integração de repositórios de dados à plataforma PowerHub, o que, em princípio, permitiria a integração de dados disponibilizados por aplicações e sistemas desenvolvidos pela Petrobras. Contudo, a API para acesso aos repositórios consiste apenas de uma interface de *query* para acesso a bases de dados, baseada em Java JDBC; essa API não oferece acesso a *bulk data*, como volumes sísmicos. Para acessar esse tipo de dado, é necessário que as aplicações implementem seus próprios mecanismos, baseados em *toolkits* providos pelos fornecedores das aplicações que mantém os repositórios.

Uma outra restrição com respeito ao uso da plataforma PowerHub como solução de integração de aplicações e dados de E&P é a imposição do uso da linguagem Java para a conexão de aplicações ao PowerHub. Essa restrição impõe dificuldades relevantes para a integração de diversas aplicações desenvolvidas pela Petrobras, como o SIGEO e o v3o2.

Finalmente, a plataforma PowerHub contempla apenas o acesso integrado a diferentes repositórios de dados; facilidades para a colaboração entre aplicações não são oferecidas por essa plataforma.

2.2 Tecnologias para desenvolvimento de uma solução própria

Discutir soluções de integração de aplicações sem fazer referência ao conceito de arquitetura orientada a serviços (*Service-Oriented Architecture*, ou SOA) é, hoje, praticamente impossível. Apesar de não ser um conceito realmente novo, os benefícios obtidos pela adoção dos princípios de SOA para a integração de aplicações em ambientes

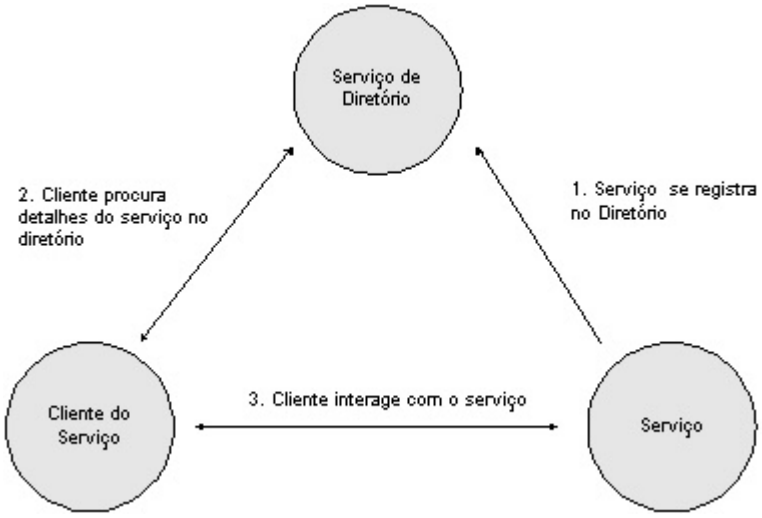

Figura 1: Arquitetura orientada a serviços

heterogêneos são cada vez mais reconhecidos no mercado, tornando essa arquitetura um padrão de referência para a implementação de soluções de integração.

Numa arquitetura SOA, aplicações compartilham dados e funcionalidades sob a forma de serviços e interagem atuando como consumidores e/ou provedores desses serviços. A principal característica desse tipo de arquitetura é o baixo grau de acoplamento entre provedores e consumidores de serviços. Para atendê-la, os serviços oferecidos devem ser descritos através de uma linguagem independente de plataforma e linguagem de programação, isolando o consumidor de um serviço dos detalhes específicos de sua implementação. Esse tipo de descrição "neutra" permite, por exemplo, que consumidores e provedores de serviço sejam desenvolvidos em linguagens diferentes, e executem sob diversas plataformas. Além disso, a disponibilização de um serviço de registro, ou diretório, permite que servidores publiquem suas ofertas de serviço, e que consumidores desses serviços possam pesquisá-las, conhecer seus detalhes e interagir com os serviços, sem conhecer, de antemão, a identidade ou localização de seus provedores. Essa relação dinâmica entre provedores e consumidores de serviço, ilustrada na Figura 1, confere ao ambiente um alto grau de reuso, extensibilidade, e de adaptação a mudanças.

A implementação de uma arquitetura SOA não pressupõe o uso de uma tecnologia específica; qualquer tecnologia ou infraestrutura que dê suporte à descoberta dinâmica de serviços e à descrição desses serviços com base em interfaces bem definidas e independentes de implementação pode ser utilizada. Contudo, a adoção de padrões abertos para a descrição e publicação de serviços e para a interação entre consumi-

dores e provedores de serviços é fundamental para garantir a interoperabilidade em ambientes heterogêneos e a independência de fornecedores de tecnologias. Nesse sentido, tanto serviços WEB como a própria arquitetura CORBA, tecnologias baseadas em padrões abertos, independentes de fornecedor e de ampla aceitação no mercado, são opções a serem consideradas para a implementação de uma solução de integração multi-plataforma. Uma outra opção a ser considerada é Java Business Integration (JBI), um padrão para a implementação de uma arquitetura SOA recentemente proposto pela Sun à comunidade Java.

2.2.1 Serviços WEB

Serviços WEB [3] são, basicamente, envelopes (*wrappers*) XML que encapsulam diversos tipos de sistemas de software (objetos distribuídos, sistemas de bancos de dados, aplicações WEB, aplicações convencionais), oferecendo formas padronizadas para acesso aos dados e outras funcionalidades desses sistemas. O acesso a um serviço WEB é realizado através do envio de mensagens de requisição de serviço representadas por documentos XML; é responsabilidade da implementação da interface de um serviço WEB transformar essas mensagens XML para o formato apropriado à invocação do sistema ou componente encapsulado e, quando for o caso, criar a mensagem XML correspondente à resposta de uma requisição.

A tecnologia básica para a construção de serviços WEB é, portanto, XML — a linguagem é utilizada para descrever todo o tipo de dado, ou informação, envolvido na interação com esses serviços. Além de XML, uma infraestrutura de integração baseada em serviços WEB envolve as seguintes tecnologias relacionadas:

- WSDL (Web Services Description Language)

  É a linguagem utilizada para descrever a interface oferecida por um serviço WEB. Essa descrição define os tipos de dados trocados entre consumidores e serviços, as operações realizadas sobre esses dados e os mapeamentos (*bindings*) para os protocolos utilizados no transporte das mensagens.

  De forma geral, o mapeamento das funcionalidades de um determinado sistema para uma interface de serviço WSDL não é uma tarefa trivial; a descrição WSDL de um serviço é significativamente mais complexa que a interface "nativa" desse serviço. Tipicamente, fornecedores de sistemas de banco de dados, de infraestruturas de *middleware*, e de infraestruturas de integração baseadas em serviços WEB provêem ferramentas que automatizam tanto a geração das descrições de serviço WSDL como também a geração das implementações dessas interfaces

(i.e, o mapeamento das interações descritas pela interface WSDL para as invocações do componente encapsulado pelo serviço WEB).

- SOAP (Simple Object Access Protocol)

  É o protocolo utilizado para a troca de mensagens entre consumidores e serviços WEB. Mensagens SOAP são documentos XML transportadas geralmente em requisições e respostas HTTP; contudo, a definição do protocolo SOAP admite o uso de outros tipos de transporte.

- UDDI (Universal Description, Discovery and Integration)

  É o serviço de registro utilizado para a publicação dos serviços WEB oferecidos em um ambiente de integração. A especificação do *framework* UDDI define um modelo de dados em XML para o armazenamento de informações sobre serviços — incluindo a localização de sua descrição em WSDL — e APIs SOAP para a publicação e consulta dessas informações.

A tecnologia de serviços WEB dá suporte a dois padrões básicos de interação: baseada em chamada remota de procedimentos (RPC) e orientada a documento. Numa interação baseada em RPC, a mensagem XML enviada pelo consumidor de um serviço representa uma chamada de método associada a parâmetros de entrada e saída, e produz uma resposta síncrona para esse consumidor. Esse tipo de interação é geralmente observado quando sistemas de objetos distribuídos (como CORBA ou J2EE) ou outros tipos de sistemas com interfaces "nativas" baseadas em RPC têm suas interfaces mapeadas diretamente para interfaces WSDL. Esse mapeamento, contudo, não expõe toda a funcionalidade provida por um sistema de objetos distribuídos [7, 9]. A tecnologia de serviços WEB, por exemplo, não inclui a noção de referências a objetos, essencial a padrões de interação que requeiram a manutenção de estado entre requisições de serviço. Apesar de ser possível expor objetos como serviços WEB, representando-os através de URIs (*Universal Resource Identifiers*), requisitos mais rígidos de escalabilidade e desempenho podem inviabilizar esse tipo de solução.

No padrão orientado a documento, a interação entre um consumidor e um serviço é assíncrona; requisições de serviço são mensagens XML auto-contidas e não envolvem uma resposta imediata ao consumidor. Esse tipo de interação é similar ao implementado, por exemplo, por sistemas de mensageria (*message-based systems*), e favorece um nível de integração de granularidade significativamente mais alto que o anterior.

A adoção da tecnologia de serviços WEB pode não ser adequada em alguns cenários de integração, como o compartilhamento de dados de grande volume — dados

sísmicos, por exemplo. A representação desses dados em XML impõe uma sobrecarga relevante de processamento a consumidores e provedores de dados, além de um alto consumo da banda de comunicação, o que pode prejudicar o desempenho das aplicações e do ambiente de integração como um todo. Além disso, dependendo do tamanho do dado acessado, copiá-lo integralmente para clientes como aplicações de visualização pode ser inviável. Para atender esse tipo de cenário, serviços de acesso a repositórios usualmente disponibilizam fábricas de objetos que implementam diferentes métodos para o acesso estruturado aos dados. Acessando esses objetos, o consumidor pode obter segmentos, ou porções, do dado, à medida que forem necessários. Conforme discutimos anteriormente, essa funcionalidade dificilmente é mapeada para interfaces de serviços WEB.

De forma geral, a tecnologia de serviços WEB é mais adequada a ambientes de alto nível de integração, onde a interação entre consumidores e provedores de serviços é inerentemente assíncrona, interfaces de serviço provêem funcionalidades complexas e desempenho de comunicação não é um requisito essencial [6]. Essa propriedade não contempla todas as características e requisitos para um ambiente de integração de aplicações e dados do E&P da Petrobras.

2.2.2 Java Business Integration

Java Business Integration (JBI) [4] é uma proposta para uma arquitetura de uma infraestrutura de integração baseada em Java e em princípios de SOA. Essa arquitetura, ilustrada na Figura 2, consiste, basicamente, de um *container* (*JBI Environment*) que hospeda componentes dinamicamente conectados a um roteador de mensagens (*Normalized Message Router*). Esses componentes atuam no ambiente JBI como provedores ou consumidores de serviços, ou simultaneamente nos dois papéis. Dois tipos distintos de componentes são definidos:

- *Service Engines*

  São componentes que provêem ou consomem serviços internamente ao ambiente JBI. Esse tipo de componente é responsável pela integração de aplicações e sistemas implementados em Java e de outros recursos acessíveis através de APIs dessa linguagem, como, por exemplo, sistemas de banco de dados.

- *Binding Components*

  São componentes responsáveis pela integração de provedores e consumidores de serviços externos ao ambiente JBI — aplicações ou sistemas implementados

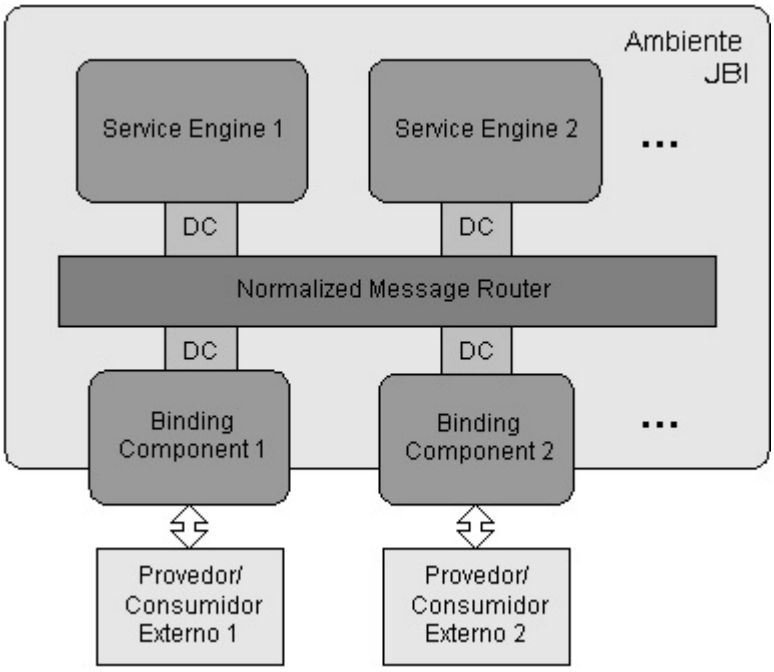

Figura 2: Arquitetura JBI

em outras linguagens, ou sobre outras plataformas. Um *Binding Component* atua, basicamente, como um representante do componente externo, traduzindo os protocolos e formatos utilizados na interface com esse componente para o padrão adotado no ambiente JBI.

Assim como em uma infraestrutura baseada em serviços WEB. os componentes de um ambiente JBI interagem através de um mecanismo de troca de mensagens baseado em XML, e descrevem seus serviços através de interfaces WSDL. A interface de um serviço compreende um conjunto de funções — operações WSDL — que, dependendo do modelo de interação especificado (*One-Way* ou *Request-Response*), podem ou não produzir respostas síncronas ao consumidor do serviço.

A interação entre componentes JBI é mediada por um roteador de mensagens — *Normalized Message Router* (NMR). É esse roteador quem recebe as mensagens geradas por um consumidor ou provedor de serviço e as direciona ao componente apropriado ao seu processamento. Além de intermediar a troca de mensagens, o NMR cumpre também o papel de serviço de registro; é através dele que provedores de serviço — *Service Engines* e *Binding Components* — publicam suas interfaces WSDL e que consumidores obtém as informações necessárias para localizar e interagir com esses serviços.

Além da infraestrutura de suporte à interação entre componentes, o padrão JBI define também mecanismos para a administração do ambiente de integração, baseados em JMX (*Java Management Extensions*). Esses mecanismos provêem suporte para a instalação de componentes, para o controle de seu ciclo de vida e para o gerenciamento do ambiente como um todo.

A arquitetura JBI é fortemente baseada na tecnologia de serviços WEB e, como essa tecnologia, privilegia ambientes de um alto nível de integração, mostrando-se inadequada para os mesmos cenários que discutimos anteriormente: o compartilhamento de dados binários e de grande volume e interações que requeiram suporte à manutenção de estado entre requisições de serviço.

Uma outra questão relevante é o foco da arquitetura JBI na integração de aplicações e sistemas baseados em Java. Atualmente, grande parte das aplicações e sistemas utilizados na área de E&P da Petrobras — tanto aplicações próprias como aplicações de mercado — é implementada em outras linguagens. A integração dessas aplicações a um ambiente JBI requer a implementação de *Binding Components*, de mecanismos de acesso para a comunicação entre as aplicações e esses componentes e de mecanismos de tradução entre as interfaces internas e externas ao ambiente JBI.

2.2.3 Arquitetura CORBA

A arquitetura CORBA [2] foi criada, em meados da década de 90, em resposta a uma demanda crescente por uma infraestrutura para o desenvolvimento de aplicações distribuídas em ambientes heterogêneos. Até o início dos anos 2000, o desenvolvimento e evolução de CORBA recebeu um enorme suporte da indústria, refletido nas centenas de membros da OMG — a organização responsável pela especificação dos padrões adotados para essa arquitetura. Diversas implementações comerciais de CORBA foram oferecidas e utilizadas para o desenvolvimento de aplicações e sistemas distribuídos em diferentes domínios, como comércio eletrônico, telecomunicações, aplicações bancárias e científicas e sistemas de tempo real. Implementações abertas, bastante completas e de qualidade — como MICO e JacORB — também contribuíram fortemente para o sucesso dessa arquitetura.

Nos últimos anos observa-se, porém, uma redução no uso de CORBA e o redirecionamento de parte do mercado para outras tecnologias. Esse redirecionamento deve-se, em grande parte, a dois fatores principais. O primeiro deles é o uso crescente da linguagem Java e de suas plataformas próprias para a construção de sistemas distribuídos, como J2EE. Entretanto, plataformas centradas em uma linguagem não são adequadas

a ambientes predominantemente heterogêneos; a integração de aplicações desenvolvidas em outras linguagens requer a implementação de adaptadores, o que implica um maior esforço de desenvolvimento e, usualmente, prejuízos de desempenho.

O segundo fator foi a introdução da tecnologia de serviços WEB e sua identificação com os benefícios de SOA. Apesar de sua adequação a alguns cenários de integração, essa tecnologia, como discutimos anteriormente, não oferece as mesmas características e funcionalidades de um sistema de objetos distribuídos, muitas delas necessárias à implementação de uma infraestrutura de integração com requisitos específicos, como é o caso da integração de aplicações de E&P.

A redução observada no uso de CORBA é natural se considerarmos que esta tecnologia existe e vem sendo trabalhada há mais de uma década. É um comportamento típico de grandes parcelas do mercado—os chamados *early adopters* e *early majority adopters*—a adoção de novas tecnologias, sem que isso represente uma real necessidade, mas movidos pelo sentimento de estarem "atualizados" tecnologicamente. Em contra partida, outros segmentos do mercado, os *late adopters*, adotam esta tecnologia atualmente, principalmente por verem em CORBA uma tecnologia madura, comprovadamente estável e eficiente. Em resumo, CORBA ainda é uma opção viável e segura para a construção de sistemas distribuídos [8].

CORBA é, basicamente, uma tecnologia de objetos distribuídos cuja principal característica é a independência de plataforma e linguagem de programação. Essa independência é garantida através do uso de uma linguagem neutra (IDL) para a descrição das interfaces de serviço oferecidas por um objeto — isto é, as operações disponibilizadas e os tipos de dados que representam os parâmetros de entrada e saída definidos para essas operações. Interfaces de serviço descritas em IDL são significativamente mais claras e sucintas que interfaces WSDL, permitindo, por exemplo, uma melhor compreensão das interações entre consumidores e provedores de um determinado serviço.

O padrão CORBA define o mapeamento de IDL para as principais linguagens de programação utilizadas atualmente, como C, C++, Java e Python, dentre outras. Implementações de CORBA tipicamente oferecem compiladores que traduzem interfaces IDL para linguagens específicas, produzindo o código que permite que clientes e provedores de serviço desenvolvidos nessas linguagens possam invocar e atender as operações definidas em IDL. Como os mapeamentos de IDL são padronizados, implementações de clientes e servidores independem do compilador de um fornecedor específico, e podem ser portadas sem qualquer alteração.

Para garantir a interoperabilidade entre diferentes implementações da arquitetura

CORBA, o transporte de requisições de serviço e de suas respostas utiliza um protocolo padronizado pela OMG (IIOP). A especificação desse protocolo define o formato das mensagens trocadas, a representação dos dados transportados nessas mensagens, e o mapeamento da comunicação sobre conexões TCP/IP[1]. O projeto desse protocolo levou em consideração uma série de requisitos, dentre eles escalabilidade e desempenho. Ao contrário de SOAP, por exemplo, que utiliza XML (uma representação textual), o protocolo IIOP utiliza uma representação mais adequada ao transporte eficiente de diversos tipos de dados, inclusive dados binários. Essa característica, aliada à disponibilidade de implementações de CORBA de grande desempenho, explica o sucesso e a predominância dessa arquitetura em domínios como sistemas de tempo real e sistemas embutidos.

Além da infraestrutura para suporte à interação entre clientes e provedores de serviço, o padrão CORBA define também uma série de serviços básicos, dentre eles um serviço de nomes e um serviço de *trading*

A função básica do serviço de nomes é permitir que clientes de um serviço obtenham dinamicamente a localização — isto é, a referência — dos objetos que constituem os pontos de acesso a esse serviço. Para prover essa funcionalidade, o serviço de nomes mantém um repositório que armazena associações (*bindings*) entre nomes e referências de objetos. Provedores de serviços fornecem ao serviço de nomes essas associações; clientes de serviços utilizam o serviço de nomes para obter a localização do objeto associado a um determinado nome.

O serviço de *trading* oferece uma funcionalidade mais rica para a descoberta de provedores de serviços, permitindo a associação de características, ou propriedades, às ofertas de um tipo de serviço (uma interface). Ao exportar uma oferta de serviço, um provedor especifica, além de sua interface e localização, valores para essas propriedades. Através de consultas ao serviço de *trading*, consumidores de um serviço podem obter a localização de um ou mais provedores que satisfaçam determinados critérios — restrições ou condições quanto aos valores das propriedades.

A especificação de interfaces de serviço em um linguagem neutra (IDL) e a possibilidade de descoberta dinâmica de provedores de serviço, seja através do serviço de nomes ou do serviço de *trading*, provêem o suporte necessário para a implementação de uma arquitetura SOA baseada em CORBA. CORBA provê também todas as facilidades de uma tecnologia de objetos distribuídos: o suporte a referências a objetos,

[1] IIOP é, na verdade, a especialização de um protocolo genérico (GIOP), que não especifica um protocolo de transporte específico, apenas um conjunto de requisitos, como o suporte a conexões bidirecionais.

à instanciação de objetos a partir de fábricas e à manutenção de estado ao longo de interações entre consumidores e provedores de serviço. Essas características, aliadas a um transporte mais eficiente de dados binários e de grande volume, permitem que uma infraestrutura baseada em CORBA atenda aos requisitos necessários à integração de aplicações de E&P.

O padrão CORBA define também dois serviços que provêem suporte à troca assíncrona de mensagens genéricas: o serviço de eventos e o serviço de notificações. Esses serviços disponibilizam canais de comunicação que atuam como roteadores, intermediando a interação entre geradores e consumidores de mensagens. Geradores e receptores de mensagens são fortemente desacoplados. Um gerador envia suas mensagens a um canal de comunicação, sem necessariamente conhecer a identidade ou localização de possíveis consumidores. Consumidores de mensagens, por sua vez, recebem mensagens de um canal sem necessariamente conhecer a identidade ou localização de seus geradores. O estabelecimento de um canal de comunicação não exige uma correspondência de um para um entre geradores e consumidores de mensagens; a conexão de múltiplos geradores e de múltiplos consumidores a um mesmo canal de comunicação é permitida.

O serviço de notificações estende a funcionalidade básica do serviço de eventos — a troca assíncrona de mensagens genéricas — oferecendo algumas facilidades adicionais, como, por exemplo, a definição de eventos estruturados, a possibilidade de um consumidor especificar filtros para restringir o tipo de eventos que deseja receber, e a configuração de diversas propriedades de qualidade de serviço — como garantia de entrega, prioridade e tempo de vida de mensagens.

O suporte à troca de mensagens assíncronas entre aplicações é uma funcionalidade importante para ambientes de integração. Além de permitir, quando adequado, um maior desacoplamento entre provedores e consumidores de serviço, essa funcionalidade é essencial a aplicações que desejam implementar facilidades de colaboração.

3 Proposta

Conforme discutimos na seção anterior, duas ferramentas de mercado objetivam a integração de aplicações da área de E&P, OpenSpirit e PowerHub. Entretanto, nenhuma das duas pode ser adotada como ferramenta de integração das aplicações de E&P da Petrobras. No caso do OpenSpirit, a razão técnica que inviabiliza sua adoção é o fato desta ferramenta não permitir a criação de um provedor de dados, o que significa que

não é possível disponibilizar em seu barramento os dados das várias bases de dados e aplicações próprias da Petrobras. Já no caso do PowerHub, há duas questões técnicas impeditivas: o fato desta ferramenta não prover suporte para a troca de alguns tipos de dados—que é o objetivo principal da integração desejada—e o fato de estar restrita à linguagem Java.

Pelo lado não técnico, adotar uma ferramenta de mercado como infra-estrutura de integração das bases de dados e das aplicações de E&P também pode não ser desejável. Usar uma ferramenta de mercado significa criar uma dependência significativa, no sentido de que todas as aplicações que forem integradas passam a conter código específico dessa ferramenta. Mais do que isso, todo o fluxo de trabalho dos profissionais passa a depender do uso dessa infra-estrutura. Assim, abandonar essa ferramenta no futuro significaria despender um grande esforço, tanto na atualização das aplicações quanto em função do comprometimento dos fluxos de trabalho. Motivos possíveis para uma eventual interrupção do uso são o aumento dos custos das licenças, a descontinuidade da ferramenta por parte do fabricante ou mesmo a inadequação, tendo em vista que o fabricante só incorpora funcionalidades que julga conveniente ao produto. Como exemplos de inadequação, podemos citar o fato do OpenSpirit não permitir a colaboração entre aplicações de diferentes usuários e a possibilidade de não contemplar um determinado tipo de dado necessário ao ambiente da Petrobras.

Em função dessas questões, nós propomos a construção de uma nova arquitetura de integração de aplicações, baseada em um barramento de serviços, própria da Petrobras, que chamamos de OpenBus. Com uma arquitetura própria, todos os requisitos relevantes do ambiente da Petrobras poderão ser contemplados. Além disso, a arquitetura poderá ser mantida e estendida, de modo a contemplar novos requisitos que venham a surgir, se adequando às necessidades futuras.

O desenvolvimento de uma arquitetura de integração e sua adoção nas aplicações representa uma tarefa de esforço considerável. A infra-estrutura, envolvendo o servidor e os serviços básicos a serem providos, é complexa e a construção das pontes que ligam cada aplicação ao barramento envolvem conhecimentos específicos destas aplicações. Em função do esforço não ser pequeno, é importante que o ponto de partida deste trabalho seja:

1. O levantamento dos requisitos a serem contemplados pela arquitetura.
2. A definição das tecnologias a serem empregadas na implementação.

Na elaboração desta proposta, definimos os seguintes requisitos para o OpenBus:

Arquitetura aberta e extensível a diferentes domínios. O E&P abrange várias áreas de conhecimento além de geologia e geofísica. Além disso, considerando a empresa inteira, há inúmeras áreas nas quais a existência de uma arquitetura de integração de aplicações seria de grande valia. Assim, um dos requisitos do OpenBus é que ele seja flexível para atender diferentes domínios. Por exemplo, deve ser possível montar um integrador de aplicações de engenharia reaproveitando toda a infra-estrutura e os serviços básicos que serão desenvolvidos inicialmente para aplicar em geologia e geofísica.

Dados estruturados e adequados ao domínio. É importante que os dados sejam modelados de forma estruturada e de acordo com o domínio para permitir que as aplicações naveguem por estes dados, encontrem e recuperem as informações desejadas. Esse requisito é fundamental para permitirmos que dados de grandes volumes possam ser utilizados. Por exemplo, um volume sísmico deve ser visto conforme sua estrutura lógica, dividido em atributos, *inlines* e *crosslines*, ao invés de uma mera seqüência de bytes, como armazenado em um arquivo no formato *seg-y*.

Compatível com múltiplas linguagens. Para integrar de forma nativa e, em decorrência, eficiente as aplicações próprias da Petrobras, é necessário que o OpenBus seja independente de linguagem de programação, pois há várias aplicações escritas em C, C++, Java, .NET etc. Deve ser possível conectar todas essas aplicações ao barramento.

Facilidade para criação de clientes. A exemplo do que é feito pelo OpenSpirit, é interessante prover bibliotecas, nas principais linguagens de desenvolvimento, que facilitem a construção de pontes ligando aplicações ao barramento. Esse requisito é importante para permitir que os próprios programadores das aplicações possam conectá-las ao ambiente integrador.

Facilidade para criação de servidores de dados. É fundamental que possamos disponibilizar no barramento os dados de qualquer banco ou aplicação própria da Petrobras. Assim viabilizamos o maior grau de integração possível entre as diferentes aplicações.

Troca de dados e mensagens entre as aplicações. A troca de dados entre as aplicações é o objetivo principal da arquitetura de integração. Porém, a troca de mensagens genéricas, de forma síncrona e assíncrona, permite aumentar significativamente as possibilidades de integração entre essas aplicações.

Eficiência da troca de grandes volumes de dados. Os dados sísmicos são da ordem de grandeza de centenas de Gigabytes e precisam ser trocados entre as aplicações

dos domínios de geologia e geofísica. A transferência desses dados não devem apenas ser permitida mas deve ser o mais eficiente possível.

Escalabilidade na troca de grandes volumes de mensagens. A troca de mensagens, conforme apresentamos, flexibiliza as funcionalidades decorrentes da integração. Por exemplo, através da troca de mensagens, duas aplicações podem sincronizar alterações simultâneas sendo feitas em um mesmo dado que seja visualizado em ambas. Esse tipo de uso pode exigir grandes volumes de mensagens e a arquitetura deve dar suporte, sem degradação de desempenho do sistema.

Suporte a mecanismos de autenticação e autorização. Em praticamente qualquer área da Petrobras, a segurança da informação é essencial. O OpenBus deve, então, dar suporte para que a autenticação dos usuários e a autorização de acesso a dados e serviços sejam feitos conforme necessário.

Suporte ao trabalho colaborativo. A integração de aplicações para a troca de dados, aliada à troca de mensagens que citamos acima formam uma base para o trabalho colaborativo entre usuários. A arquitetura pode, então, oferecer os serviços básicos, no nível de abstração adequado, para as aplicações que desejem explorar a colaboração.

Especificados os requisitos, passamos à definição das tecnologias a serem utilizadas na implementação do OpenBus. A construção de um barramento comum através do qual aplicações possam se conectar para trocar dados e mensagens, como descrevemos, já sugere uma abordagem baseada em serviços, ou um SOA. Uma arquitetura SOA atende à solução que propomos por garantir um alto grau de desacoplamento entre os membros do barramento, viabilizando a integração de aplicações heterogêneas, e por não implicar em uma tecnologia específica para sua implementação, nos permitindo adotar uma que atenda aos requisitos funcionais, ainda considerando desempenho e escalabilidade. Além de atender aos requisitos técnicos específicos, a adoção de uma arquitetura orientada a serviços também atende às diretrizes estabelecidas pela TI-IDTA-AT em [1].

Com base nos requisitos levantados, a tecnologia que julgamos mais adequada à implementação do OpenBus é CORBA. Como apresentamos na seção anterior, CORBA possui todos os requisitos para uma implementação de SOA. Além disso, CORBA oferece suporte para descrição clara e sucinta das interfaces dos serviços, à construção de componentes em múltiplas linguagens, à mensageria, à implementação de mecanismos de autenticação e ao acesso, via referências, a objetos remotos que podem preservar estado. Por fim, CORBA possibilita um desempenho muito superior quando comparado com as tecnologias que adotam comunicação baseada em XML.

Na próxima seção, nós apresentamos a arquitetura proposta para o OpenBus, descrevendo suas macro-funcionalidades e os serviços básicos que estarão disponíveis para as aplicações.

4 Arquitetura

O OpenBus é baseado no *middleware* CORBA e em alguns dos serviços padronizados para sua arquitetura. Seu barramento é formado por componentes que podem oferecer serviços ou podem apenas consultar serviços de outros componentes. A infraestrutura do OpenBus provê alguns serviços básicos que podem ser oferecidos por um único componente ou podem estar distribuídos em diversos componentes em diferentes máquinas. A medida que as aplicações se conectam ao barramento, o espaço de serviços que o OpenBus oferece cresce. Desta forma, o OpenBus pode ser estendido com serviços para acesso a repositórios de dados (Base integrada, OpenSpirit), acesso a dados de uma aplicação (Websintesi, v3o2), etc. (Figura 3.)

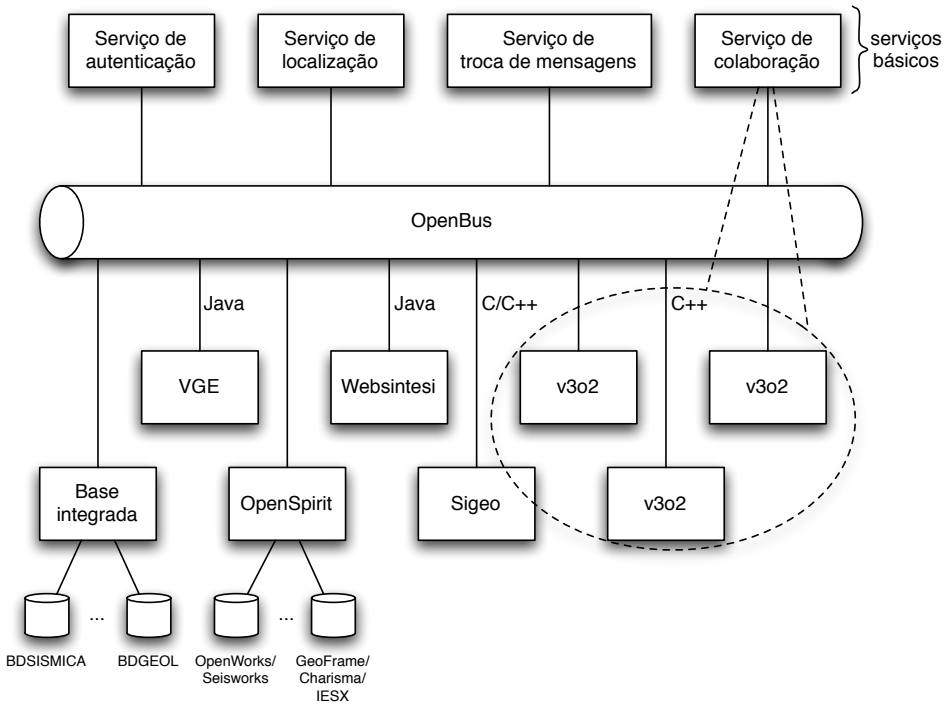

Figura 3: Arquitetura do OpenBus

Qualquer aplicação que queira fazer parte do barramento deve implementar um componente para o OpenBus. Através desse componente é possível acessar todos os serviços existentes no barramento. Para isso, o componente deve ser capaz de se conectar ao barramento e se autenticar. Através dos serviços oferecidos pelo OpenBus, o componente pode acessar um serviço previamente conhecido, ou pode consultar o serviço de localização para descobrir qual componente oferece o serviço que ele deseja acessar. Para oferecer um serviço, uma aplicação pode usar o mesmo componente ou implementar outro. Tal componente deve ser capaz de publicar no barramento a interface do serviço que deseja oferecer.

Entre os serviços básicos oferecidos pelo OpenBus podemos destacar os serviços de autenticação, localização, troca de mensagens e colaboração.

- Serviço de autenticação

  Para fazer parte do barramento, um componente deve se autenticar a ele. Existem tipicamente duas formas de autenticação para um componente. Uma onde o componente usa a credencial de um usuário e outra onde o componente (ou aplicação) possui uma credencial própria.

  Aplicações como o VGE, onde o usuário fornece seu identificador e sua senha, podem usar a própria credencial do usuário para autenticar seus componentes no barramento. Já as aplicações que executam como um *daemon* podem utilizar uma credencial própria.

  Como apenas um componente do barramento pode acessar serviços de outro componente e todos os componentes do barramento já foram autenticados, os componentes do barramento são considerados confiáveis. Entretanto, cabe a cada serviço verificar se um componente possui a devida autorização para acessar seus serviços.

- Serviço de localização

  O serviço de localização é baseado no serviço de *trading* de CORBA. Este serviço facilita a oferta e o descobrimento de instâncias de serviços de tipos específicos. Com este serviço, componentes podem publicar suas características e outros componentes podem encontrar os serviços desejados através das características publicadas.

  Para publicar um serviço, um componente provê ao serviço de localização uma descrição de seu serviço e a sua localização. Para encontrar um serviço, um componente pergunta ao serviço de localização se existe algum serviço com determinadas características. O serviço de localização procura entre as descrições

que possui e responde com a localização da interface do serviço selecionado. O componente que solicitou o serviço pode então acessá-lo.

- Serviço de troca de mensagens

  O serviço de troca de mensagens é baseado no serviço de notificação de CORBA. Através dele, as aplicações podem comunicar-se livremente, trocando mensagens diretas ou registrando interesse em receber mensagens quando algum evento ocorre na outra aplicação.

  Quando duas aplicações estão vendo o mesmo dado e uma delas o altera, através desse serviço, ela pode enviar uma mensagem para a outra aplicação para que ela atualize a sua cópia do dado.

- Serviço de colaboração

  O serviço de colaboração usa o serviço de troca de mensagens. Através dele, um componente pode criar uma sessão de colaboração e definir quem são os componentes participantes. Inicialmente, cada componente é convidado pelo serviço de colaboração a fazer parte desta sessão.

  Cada sessão de colaboração possui atributos específicos e pode envolver mais de um tipo de aplicação de diferentes usuários. Por exemplo, uma colaboração entre diferentes instâncias do v3o2 pode ter atributos de câmera, para que as diferentes instâncias tenham a mesma visão do volume (Figura 3.) Através de uma sessão, os componentes envolvidos podem compartilhar diferentes tipos de eventos, como por exemplo, eventos de interação de usuário (seleção e alteração de dados, movimentação de cursor). Cada componente pode gerar e receber esses eventos.

5 Conclusões

A integração entre as aplicações da área de E&P da Petrobras e, em especial, entre essas aplicações e as bases de dados corporativas é uma necessidade clara para os fluxos de trabalho atuais. Entre as diferentes formas de estabelecer esta integração, a mais adequada, em nossa opinião, é através de um barramento comum de serviços, o que ainda se mostra alinhado com as diretrizes corporativas da Companhia. Através do barramento de serviços, não apenas alcançamos a integração básica, por cópia de dados, mas também viabilizamos, na mesma infra-estrutura, formas mais ricas de integração, como a monitoração de dados compartilhados e mecanismos de colaboração.

As ferramentas disponíveis no mercado, voltadas a este fim, não atendem aos rígidos requisitos que a área de E&P estabelece. Em função disso, propomos a construção de uma arquitetura de integração de aplicações, baseada em um barramento de serviços, que chamamos de OpenBus.

Após uma análise das tecnologias disponíveis atualmente para a construção de uma arquitetura de serviços, concluímos que a arquitetura OpenBus deve ser implementada sobre o *middleware* CORBA. Essa escolha se justifica pelo fato de que CORBA oferece a forma mais adequada para implementar uma arquitetura de serviços cujos componentes sejam heterogêneos e os requisitos de escalabilidade e desempenho sejam críticos. Em função desses requisitos, e do contexto mais amplo da Petrobras, definimos que a arquitetura OpenBus deverá:

- Ser aberta e extensível a domínios fora do E&P.
- Permitir a troca de dados estruturados de forma adequada ao domínio.
- Permitir a integração de aplicações escritas em diferentes linguagens.
- Facilitar a integração de provedores e consumidores de dados.
- Permitir a troca de mensagens genéricas entre aplicações.
- Apresentar eficiência na troca de dados de grande volume.
- Apresentar escalabilidade na troca de grandes volumes de mensagens.
- Prover suporte a mecanismos de autenticação e autorização.
- Prover suporte para a implementação de colaboração.

A implementação da arquitetura deverá prover os serviços básicos de autenticação, localização, mensageria e colaboração de forma operacional, para que esta estrutura possa ser reutilizada em diferentes contextos da Companhia. Dentro do foco no E&P, esta estrutura será especializada para modelar os dados dos domínios de geologia e geofísica. Por fim, serão codificadas as pontes de ligação da Base Integrada e das aplicações WebSintesi, VGE, v3o2 e Sigeo que os conectarão ao barramento, permitindo sua integração e servindo de prova de conceito para a arquitetura OpenBus.

Referências

[1] M. J. Amaral. Integração e Interoperabilidade - Relatório Final de Prospecção, Tecnologia da Informação, Petrobras, 2006. TI - IDTA - AT.

[2] F.Bolton. *Pure CORBA*. Sams Publishing, 2002.

[3] E. Newcomer. *Understanding Web Services: XML, WSDL, SOAP and UDDI*. Addison-Wesley, 2002.

[4] R. Ten-Hove and P.Walker. JSR 208: Java Business Integration (JBI) 1.0, August 2005. `http://www.jcp.org/en/jsr/detail?id=208`.

[5] The OpenSpirit Corporation. OpenSpirit - Products, 2006. `http://www.openspirit.com/products.html`.

[6] S. Vinoski. Putting the "Web" into Web Services: Web Services Interaction Models, Part 2. *IEEE Internet Computing*, 6(4):90–92, July/August 2002.

[7] S. Vinoski. Web Services Interaction Models – Part I. *IEEE Internet Computing*, 6(3):89–91, May/June 2002.

[8] S. Vinoski. Is Your Middleware Dead? *IEEE Internet Computing*, 8(4):94–96, September/October 2004.

[9] W. Vogels. Web services are not distributed objects. *IEEE Internet Computing*, 7(6):59–66, November/December 2003.