4 Estudo de Caso

Para avaliar a adequação do modelo proposto e verificar as diferenças de uso entre o SCS [13] original e a nossa proposta, realizamos um estudo de caso sobre uma aplicação real, utilizando o histórico de evolução de diferentes componentes SCS.

A aplicação escolhida foi o OPENBUS [30], pois esse *middleware* já possui a necessidade de prover mais de uma versão de serviços para manter compatibilidade com clientes que ainda não foram atualizados. Uma breve descrição do OPENBUS pode ser encontrada no apêndice B.

O estudo de caso foi realizado em duas etapas. Na primeira etapa nós adequamos a versão 1.4 do OPENBUS para utilizar o SCS-MV. A característica principal dessa versão é que ela foi a última versão que trabalhava apenas com uma única versão dos serviços. Na segunda etapa nós adequamos a versão 1.5 do OPENBUS, que foi a versão que passou a ter que prover diferentes versões de serviços para manter compatibilidade com clientes que não acompanharam a atualização do *middleware*. Dessa forma, neste estudo de caso, nós aplicamos o modelo em um sistema que não oferecia suporte a múltiplas versões, que foi o caso da versão 1.4, e também testamos o modelo em um sistema que precisava disponibilizar múltiplas versões de diferentes serviços, que foi o caso da versão 1.5. Vale destacar que a versão 1.5 tinha que oferecer compatibilidade retroativa exatamente com a versão 1.4.

O desenvolvimento do nosso modelo foi inspirado na análise da implementação da versão 1.5 do OPENBUS, que utilizava a abordagem tradicional de suporte a múltiplas versões. Logo, o estudo de caso avalia se esse modelo atende bem a esse caso específico. Isso não chega a ser uma avaliação completa do modelo, porque o modelo foi fortemente influenciado por esse estudo de caso. Ainda assim, esse exercício já permitiu identificar alguns pontos que poderiam ser melhorados, mas são necessários outros estudos de caso para avaliar o modelo em situações diferentes dessa, que foi a inspiradora do modelo.

Com a experiência obtida durante o estudo de caso, iremos realizar uma avaliação enumerando os tipos de modificações necessárias, assim como outras modificações que o novo modelo sugere, nas seções 4.2 e 4.3 respectivamente.

PUC-Rio - Certificação Digital Nº 0912832/CA

Na seção 4.1, apresentaremos as principais modificações que ocorreram entre as duas versões do OPENBUS.

4.1 Mudanças entre as versões do Openbus

A infra-estrutura do OPENBUS provê dois serviços básicos necessários para que o usuário possa utilizar o barramento, o serviço de acesso e o serviço de registro, e um terceiro serviço, que auxilia o compartilhamento de informações entre um grupo de clientes, denominado serviço de sessão. Nesta seção, apresentaremos as principais mudanças que aconteceram nesses três serviços principais.

4.1.1 Serviço de Control de Acesso

A primeira mudança importante foi a alteração dos módulos das interfaces desse serviço. Na versão 1.4, o módulo das interfaces era *openbusidl/acs*. Na versão 1.5, o módulo foi alterado para *tecgraf/openbus/-core/v1_05/access_control_service*, como ilustrado pelo código 4.1.

Código 4.1: Modificação no módulo das interfaces do serviço de controle de acesso.

```
1 -module openbusidl {
2 -module acs {
3 +module tecgraf {
4 +module openbus {
5 +module core {
6 +module v1_05 {
7 +module access_control_service {
```

Outras mudanças importantes, foram a atualização da interface do serviço de controle de acesso, onde tivemos a remoção e inclusão de alguns métodos (explicitados no código 4.2), e a inclusão da interface *access_control_service/IManagement*, que reflete como a inclusão de uma nova faceta do componente do serviço de controle de acesso.

Código 4.2: Modificação na interface do serviço de controle de acesso.

```
1   interface IAccessControlService {
2     ...
3 -   boolean setRegistryService(in rs::IRegistryService registryServiceComponent);
4 -   rs::IRegistryService getRegistryService ();
5 +   BooleanSeq areValid (in CredentialSeq credentials);
6 +   EntryCredential getEntryCredential (in Credential aCredential);
7 +   EntryCredentialSeq getAllEntryCredential ();
8     ...
9   }
```

PUC-Rio - Certificação Digital Nº 0912832/CA

4.1.2 Serviço de Registro

Também ocorreu uma mudança nos módulos do serviço de registro. Mudou de *openbusidl/rs* para *tecgraf/openbus/core/v1_05/registry_service*, como ilustrado pelo código 4.3.

Código 4.3: Modificação no módulo das interfaces do serviço de registro.

```
1 -module openbusidl {
2 -module rs {
3 +module tecgraf {
4 +module openbus {
5 +module core {
6 +module v1_05 {
7 +module registry_service {
```

Além disso, a interface do serviço de registro sofreu atualizações nas assinaturas de dois métodos (explicitados no código 4.4), e a inclusão da interface *registry_service/IManagement*, que difere da interface de gerência do serviço de controle de acesso.

Código 4.4: Modificação na interface do serviço de registro.

```
1     interface IRegistryService {
2 -     boolean register(in ServiceOffer aServiceOffer, out RegistryIdentifier identifier);
3 +     RegistryIdentifier register(in ServiceOffer aServiceOffer)
4 +       raises (UnathorizedFacets);
5       boolean unregister(in RegistryIdentifier identifier);
6 -     boolean update(in RegistryIdentifier identifier, in PropertyList newProperties);
7 +     void update(in RegistryIdentifier identifier, in PropertyList newProperties)
8 +       raises (UnathorizedFacets, ServiceOfferNonExistent);
9       ServiceOfferList find (in FacetList facets);
10      ServiceOfferList findByCriteria (in FacetList facets, in PropertyList criteria);
11 +    ServiceOfferEntryList localFind (in FacetList facets, in PropertyList criteria);
12    };
```

4.1.3 Serviço de Sessão

O seviço de sessão também acompanhou essa reorganização dos módulos das IDLs, mudando de *openbusidl/ss* para *tecgraf/openbus/session_service/v1_05*, como ilustrado pelo código 4.5.

Código 4.5: Modificação no módulo das interfaces do serviço de sessão.

```
1 -module openbusidl {
2 -module ss {
3 +module tecgraf {
4 +module openbus {
5 +module session_service {
6 +module v1_05 {
```

PUC-Rio - Certificação Digital Nº 0912832/CA

4.2 Modificações necessárias

Várias modificações foram necessárias para adequar as versões do OPENBUS ao novo modelo estendido. Nesta seção, iremos apresentar partes de códigos que ilustram as diferenças entre o código original e a adequação necessária, e comentar todos os tipos de modificações realizadas nas subseções enumeradas a seguir:

- Descritores de componentes,
- Acesso à faceta via contexto,
- Retorno do método *getFacet* e *getFacetByName*,
- Parâmetro do método *connect*,
- Uso da estrutura *FacetDescription*,
- Uso da estrutura *ConnectionDescription*.

Para fins de clareza e simplificação, os códigos que serão apresentados nas subseções a seguir foram ligeiramente modificados para diminuir a quantidade de informações apresentadas. As informações omitidas não são essenciais para o entendimento das modificações realizadas.

4.2.1 Descritores de componentes

O modelo de componente proposto não obriga que o desenvolvedor do componente SCS reveja todos os descritores de facetas e receptáculos que utiliza na construção dos componentes (descritos na seção 3.4.1). No caso dos descritores de faceta, se o componente não provê suporte a múltiplas versões de uma interface e não define a chave para criar o IOR persistente, nenhuma modificação é necessária. Nesse caso, o método de construção do componente (*newComponent*) criará automaticamente as interfaces de acesso da versão corrente das facetas disponibilizadas, e gerará automaticamente uma chave para o cadastro dos *servants* das facetas e das interfaces de acesso.

Se o descritor de faceta define a chave de geração do IOR, então é necessário modificá-lo. A chave do *servant*, que era definida no descritor (código 4.6, linha 9), correspondia a chave definida na interface de acesso no SCS-MV. Então, para manter a lógica de implementada no OPENBUS, a modificação necessária era mover a chave para o descritor da interface de acesso, no campo *wrapper_key* (código 4.6, linha 5). O campo *key* do descritor da faceta define a chave de cadastro do *servant* da faceta (objeto *IFacet*), mas

PUC-Rio - Certificação Digital Nº 0912832/CA

Código 4.6: Modificações do descritor da faceta *IAccessControlService* na versão 1.4 do Openbus.

```
1   local iACS = "IDL:openbusidl/acs/IAccessControlService:1.0"
2  +local acsVersions = {}
3  +acsVersions[iACS] = {}
4  +acsVersions[iACS].interface_name = iACS
5  +acsVersions[iACS].wrapper_key    = "ACS"
6   facetDescriptions.IACS.name               = name
7   facetDescriptions.IACS.interface_name     = iACS
8   facetDescriptions.IACS.class              = ACSFacet
9  -facetDescriptions.IACS.key                = "ACS"
10 +facetDescriptions.IACS.versions           = acsVersions
```

o mesmo não foi utilizado no estudo de caso, e por isso não aparece no exemplo de código mostrado.

Quando o componente dá suporte a múltiplas versões de uma interface, o seu descritor precisa ser modificado para fazer uso das abstrações de interfaces de acesso do modelo proposto. No caso do Openbus 1.5, simulou-se o suporte a múltiplas versões no modelo original do SCS adicionando novas facetas para cada versão da interface. Sendo assim, a modificação necessária nos descritores de faceta é a remoção das facetas adicionais (código 4.7, linhas 19-22) e inclusão das interfaces de acesso para cada versão suportada (código 4.7, linhas 2-11).

Código 4.7: Modificações do descritor da faceta *IAccessControlService* na versão 1.5 do Openbus.

```
1  -facetDescriptions.IACS.name = "IAccessControlService_v1_05"
2  +local acsVersions = {}
3  +-- versao 1.5
4  +acsVersions[iACS] = {}
5  +acsVersions[iACS].interface_name = iACS
6  +acsVersions[iACS].wrapper_key = iACSkey
7  +-- versao 1.4
8  +acsVersions[old-iACS] = {}
9  +acsVersions[old-iACS].interface_name = old-iACS
10 +acsVersions[old-iACS].wrapper_key = old-iACSkey
11 +acsVersions[old-iACS].wrapper_class = 1_4.ACSFacet
12 +-- faceta
13 +facetDescriptions.IACS.name               = name
14  facetDescriptions.IACS.interface_name     = iACS
15  facetDescriptions.IACS.class              = ACSFacet
16 -facetDescriptions.IACS.key                = iACSkey
17 +facetDescriptions.IACS.versions           = acsVersions
18
19 -facetDescriptions.oldIACS.name               = name
20 -facetDescriptions.oldIACS.interface_name     = old-iACS
21 -facetDescriptions.oldIACS.class              = 1_4.ACSFacet
22 -facetDescriptions.oldIACS.key                = old-iACSkey
```

Os descritores dos receptáculos do componente também precisam ser modificados, pois, como os receptáculos deixaram de estar associados a uma interface específica, a informação do nome da interface não é mais necessária (código 4.8, linha 4), e eles passaram a permitir a definição da lista de versões compatíveis através do campo *compatible_versions* (código 4.8, linha 5).

PUC-Rio - Certificação Digital Nº 0912832/CA

Código 4.8: Modificação do descritor do receptáculo do Serviço de Registro.

```
1  local recepDescs = {}
2  recepDescs.RSReceptacle = {}
3  recepDescs.RSReceptacle.name           = "RegistryServiceReceptacle"
4 -recepDescs.RSReceptacle.interface_name =  "IDL:scs/core/IComponent:1.0"
5 +recepDescs.RSReceptacle.compatible_versions = {}
6  recepDescs.RSReceptacle.is_multiplex   = true
```

No nosso estudo de caso não houve a necessidade de explicitar as versões compatíveis dos receptáculos, ou seja, definimos a lista de compatibilidade como vazia, o que significa dizer que o receptáculo aceita conectar com qualquer versão. Essa abordagem foi possível, porque os receptáculos dos componentes só são utilizados na configuração do próprio OPENBUS, e, por decisão de projeto, os componentes do OPENBUS são implementados para sempre se comunicarem na versão mais nova dos sub-serviços.

4.2.2 Acesso à faceta via contexto

O modelo de componente SCS permite que um componente acesse as implementações de suas facetas através do objeto *contexto*, e não procura restringir as flexibilidades que a linguagem LUA oferece. Sendo assim, nada impede que o desenvolvedor sobrescreva métodos dos objetos que representam as facetas e que acesse seus atributos internos (privados em outras linguagens).

O código 4.9 ilustra uma mudança necessária para sobrescrever métodos da interface *IComponent* após a instanciação do componente, e o código 4.10 mostra o acesso a atributos internos da faceta.

Código 4.9: Sobrescrevendo os métodos do componente após a inicialização.

```
1  acsInst = scs.newComponent(facetDescs, recepDescs, componentId)
2  -- Configurações
3 -acsInst.IComponent.startup = AccessControlService.startup
4 -acsInst.IComponent.shutdown = AccessControlService.shutdown
5 +acsInst.IComponent._facet_ref.startup = AccessControlService.startup
6 +acsInst.IComponent._facet_ref.shutdown = AccessControlService.shutdown
```

Código 4.10: Acesso a atributos internos (privados) da implementação da interface.

```
1  acsInst = scs.newComponent(facetDescs, recepDescs, componentId)
2 -local acs = acsInst.IAccessControlService
3 +local acs = acsInst.IAccessControlService._facet_ref
4  acs.config = AccessControlServerConfiguration
5  acs.entries = {}
6  acs.observers = {}
```

Como é possível notar nesses códigos, a modificação necessária é o acesso ao campo *_facet_ref*. Com a inclusão das interfaces de acesso, os objetos que

PUC-Rio - Certificação Digital Nº 0912832/CA

são inseridos no contexto são as interfaces de acesso das versões correntes das facetas do componente. as interfaces de acesso funcionam como *wrappers* da versão corrente e possuem uma referência para a implementação desta versão corrente. O campo *_facet_ref* é exatamente essa referência. Por isso a necessidade de incluir essa indireção nos códigos, para que a modificação ocorra na referência da implementação corrente e não nas interfaces de acesso.

4.2.3 Retorno do método getFacet e getFacetByName

Os métodos *getFacet* e *getFacetByName*, pertencentes à interface *IComponent*, também foram modificados, vide o código 4.11. O retorno do método *getFacet*, que tem como parâmetro de entrada um nome de interface, deixa de ser a referência para o objeto (*Object*) que implementa a interface da faceta, e passa a ser uma lista de facetas (interface *IFacet*) que implementam a interface buscada. Essa modificação não é relacionada à extensão do modelo do SCS para prover o suporte a múltiplas versões. Essa modificação é uma melhoria que visa tornar o método mais genérico, adequando-se melhor ao caso onde os componentes possuem mais de um tipo de implementação da mesma interface.

Código 4.11: Modificação dos métodos *getFacet* e *getFacetByName* da interface *IComponent*.

```
1   interface IComponent {
2   ...
3 − Object getFacet (in string facet_interface);
4 + Facets getFacet (in string facet_interface);
5   ...
6 − Object getFacetByName (in string facet);
7 + IFacet getFacetByName (in string facet) raises (InvalidName);
8   ...
9   };
```

PUC-Rio - Certificação Digital Nº 0912832/CA

O SCS-MV não realiza nenhum tipo de ordenação dessa lista de facetas retornadas pelo método *getFacet*. Então, no caso de uma aplicação que possua componentes com mais de uma faceta implementando a mesma interface, o desenvolvedor talvez precise definir uma regra para escolher a faceta, caso precise utilizar apenas uma. Uma alternativa para definir essa regra de escolha é redefinir o método para realizar algum tipo de ordenação da lista de facetas retornadas.

O retorno do método *getFacetByName* também deixa de ser a referência para o objeto que implementa a interface da faceta e passa a ser a referência para o *IFacet*.

Os códigos 4.12 e 4.13 apresentam as modificações necessária para adequar o código do SCS original para o SCS-MV. No exemplo do código 4.12,

acessa-se diretamente a primeira posição da lista, porque o sub-serviço em questão não tem mais de uma faceta implementando a mesma interface.

Código 4.12: Modificação no uso do método *getFacet*.

```
1 -local facet = acsComp:getFacet(interface)
2 -facet = orb:narrow(facet, interface)
3 +-- retorno de getFacet é uma lista de IFacet
4 +local facetIF = acsComp:getFacet(interface)
5 +-- acessando o primeito IFacet disponível
6 +facetIF = facetIF[1]
7 +local accessInterface = facetIF:getAccessInterface(interface)
8 +local facet = orb:narrow(accessInterface.objref, interface)
```

Código 4.13: Modificação no uso do método *getFacetByName*.

```
1 -local facet = comp:getFacetByName(name)
2 -facet = orb:narrow(facet, interface)
3 +local facetIF = comp:getFacetByName(name)
4 +local accessInterface = facetIF:getAccessInterface(interface)
5 +local facet = orb:narrow(accessInterface.objref, interface)
```

4.2.4
Parâmetro do método connect

O processo de conexão sofreu uma pequena mudança na sua especificação. O parâmetro de entrada, que representava o objeto faceta a ser conectado, deixou de ser uma referência para *Object* e passou a ser uma referência para IFacet. O código 4.14 mostra a modificação da definição do método e o código 4.15 mostra a modificação necessária em um exemplo de uso do método.

Código 4.14: Modificação da definição do método *connect*.

```
1  interface IReceptacles {
2   ...
3 - ConnectionId connect (in string receptacle, in Object obj)
4 + ConnectionId connect (in string receptacle, in IFacet obj)
5   raises (InvalidName, InvalidConnection, AlreadyConnected,
6          ExceededConnectionLimit);
7   ...
8  };
```

Código 4.15: Modificação no uso do método *connect*.

```
1 -local acs = Openbus.ft
2 +local icacs = Openbus.ic
3 +icacs = Openbus:getORB():narrow(icacs, "IDL:scs/core/IComponent:1.0")
4 +local acsIF = icacs:getFacetByName("IFaultTolerantService")
5  local ftRec = self.context.IReceptacles
6 -self.recConnId = ftRec:connect("IFaultTolerantService",acs)
7 +self.recConnId = ftRec:connect("IFaultTolerantService",acsIF)
```

PUC-Rio - Certificação Digital Nº 0912832/CA

4.2.5 Uso da estrutura FacetDescription

A estrutura que descreve uma faceta também precisou ser modificada para também representar o conceito de que uma faceta pode possuir várias interfaces de acesso. Por isso, o descritor deixa de possuir uma referência de *Object* para a faceta e passa a conter uma lista de descritores de interfaces de acesso e o nome da versão atual da implementação da faceta (*AccessInterfaceDescription*). O código 4.16 apresenta a modificação na estrutura descritora de facetas, e o código 4.17 apresenta a estrutura que descreve as interfaces de acesso.

Código 4.16: Modificação da definição da estrutura *FacetDescription*.

```
1  struct FacetDescription {
2    string name;
3    string interface_name;
4 -  Object facet_ref;
5 +  string version;
6 +  AccessInterfaceDescriptions versions;
7  };
```

Código 4.17: A estrutura *AccessInterfaceDescription*.

```
1 +struct AccessInterfaceDescription {
2 +  string interface_name;
3 +  string version;
4 +};
```

No estudo de caso do OPENBUS identificamos um pedaço da lógica da aplicação que precisou ser modificada, pois fazia uso da lista dos descritores de faceta. O Serviço de Registro recuperava a lista de descritores de facetas através da faceta *IMetaInterface* e utilizava essa lista para cadastrar os sub-serviços disponibilizados pelos componentes. Porém, como a modificação é muito ligada a lógica da aplicação, essa modificação não será apresentada aqui por motivos de clareza.

4.2.6 Uso da estrutura ConnectionDescription

Com a mudança do tipo do parâmetro passado para a conexão, vide subseção 4.2.4, o descritor da conexão acompanha essa modificação passando a possuir a referência *IFacet* da faceta conectada. Com isso, o desenvolvedor precisa adequar o uso das conexões. O código 4.18 apresenta a modificação da estrutura, e o código 4.19 mostra um exemplo da adequação no uso da conexão.

PUC-Rio - Certificação Digital Nº 0912832/CA

Código 4.18: Modificação da definição da estrutura *ConnectionDescription.*

```
1   struct ConnectionDescription {
2     ConnectionId id;
3 - Object objref;
4 + IFacet objref;
5   };
```

Código 4.19: Modificação no uso das conexões.

```
1   local conns = recept:getConnections("RegistryServiceReceptacle")
2 -local rslComp = orb:narrow(conns[1].objref, "IDL:scs/core/IComponent:1.0")
3 +local rslCompIF = conns[1].objref
4 +local rslCompAccessInterface = rslCompIF:getAccessInterface("IDL:scs/core/IComponent:1.0")
5 +local rslComp = orb:narrow(rslCompAccessInterface.objref, "IDL:scs/core/IComponent:1.0")
```

4.3 Modificações sugeridas para a aplicação

Durante a realização do estudo de caso, identificamos alguns pontos que poderiam ser modificados para tirar maior proveito do novo modelo. Como uma das premissas do estudo de caso era de apenas realizar as modificações necessárias para adequar o OPENBUS ao modelo de componente proposto, essas modificações sugeridas não foram implementadas no estudo de caso.

4.3.1 Conexão da faceta IComponent

Uma característica que chama atenção na implementação do OPENBUS é a maneira como se realiza as conexões entre os componentes do sistema. Os receptáculos servem de canal de entrada de dependências do componente, e sua figura é uma maneira de explicitar essa dependência e de modularizar o componente.

O fato interessante no OPENBUS é que a faceta conectada entre os componentes Serviço de Controle de Acesso (ACS) e Serviço de Registro (RS) é do tipo *IComponent*. Entretanto, esses componentes não dependem da faceta *IComponent* do outro. Na verdade, eles dependem de uma ou mais facetas de sub-serviços oferecidos pelo outro componente.

O que justifica a conexão da faceta *IComponent*, ao invés de outras facetas que oferecem serviços específicos da aplicação, é que o SCS original não provê um suporte para que seja possível obter a referência do componente (*IComponent*) a partir de uma faceta mais especializada. A especificação de CORBA 2.3.x oferece suporte para recuperar a referência do componente a partir de uma faceta qualquer através do método *_get_component*, mas, infelizmente, algumas implementações de ORB pararam suas implementações em versões anteriores à introdução dessa operação. Portanto, essas implementações

PUC-Rio - Certificação Digital Nº 0912832/CA

não suportam esse método. Sendo assim, não é possível garantir que o método vá existir nos lados cliente e servidor em ambientes heterogêneos (como é o caso do OPENBUS) e por isso o SCS não pode contar com este método e nem utilizá-lo na prática.

A modificação sugerida é de deixar de utilizar a faceta *IComponent* na conexão e passar a utilizar a faceta mais específica do sub-serviço desejado. Com isso, o componente expressa melhor a sua relação de dependência na sua estrutura, e, com a nova representação da faceta pela interface *IFacet*, é possível recuperar a referência do componente (a faceta *IComponent*) diretamente, descartando a necessidade de realizar a conexão pela própria faceta *IComponent*.

4.3.2
Referência para IFacet

Uma estratégia de implementação do SDK do OPENBUS é armazenar as referências para o que hoje são as interfaces de acesso dos principais sub-serviços disponibilizados pelo barramento, como o *IComponent* do componente de Serviço de Controle de Acesso, o próprio Serviço de Controle de Acesso, o Serviço de Registro, entre outros.

Dado que o *IFacet* passou a representar a faceta como um todo, essas referências mantidas pelo SDK poderiam deixar de ser as interfaces de acesso e passassem a ser os *IFacet* das facetas desejadas. O interessante desta modificação é que a referência mantida seria a mesma, mesmo que as interfaces oferecidas pelos sub-serviços mudassem, e passaria a possuir uma maneira de obter a referência do componente através da faceta (como mencionado na subseção 4.3.1). No momento de utilizar o sub-serviço bastaria escolher a interface de acesso desejada.

4.4
Considerações finais

Neste capítulo nós compilamos as modificações necessárias e sugeridas para realizar o estudo de caso do OPENBUS, adequando as versões 1.4 e 1.5 para utilizarem o modelo de componente proposto.

O primeiro ponto positivo a ser considerado é que a abstração do modelo estendido se adequou a todas as necessidades de uso do OPENBUS, e ainda propôs melhorias no uso. Essas melhorias sugeridas proporcionam uma melhor expressividade do uso da abstração nas conexões, favorecendo a conexão das facetas que o componente efetivamente depende. E, com a evolução da representação das facetas, sugere-se que os clientes guardem a referência das

PUC-Rio - Certificação Digital Nº 0912832/CA

facetas ao invés da referência das interfaces de acesso, pois isso permitiria que o cliente fizesse uso de novas interfaces de acesso sem precisar modificar a referência mantida, e porque pela faceta é possível recuperar o componente, enquanto pela interface de acesso isso não seria possível.

Um ponto negativo é que com a nova representação da faceta também introduzimos uma nova indireção. Mas não entendemos que essa indireção seja muito custosa, pois só é necessário realizar mais uma única chamada remota para obter a referência da interface de acesso desejada, não influenciando em nada na maneira de uso do mesmo. Existe uma segunda indireção adicionada no nosso modelo, que se dá pelo fato das interfaces de acesso serem *wrappers* da implementação corrente da faceta. Porém, como a interface de acesso possui uma referência local para a implementação da faceta, o custo da indireção é desprezível.

A maior contribuição que o nosso modelo proposto oferece é a extensão do modelo do SCS para manter a coesão da abstração de componentes de software e suas portas de comunicação, e, consequentemente, melhorar a modularização do sistema que se utiliza do modelo. Com as modificações realizadas na representação das facetas, com a inclusão das interfaces de acessos das facetas, com a extensão e definição do vocabulário de uso do modelo de componentes, e com a reformulação da conexão, acreditamos que demos um passo à frente para aprimorar o modelo, atentando para os conceitos de compreensibilidade, continuidade e ocultação de informação. Esses conceitos estão muito ligados à organização dos módulos para facilitar a compreensão e correção dos módulos, sem a necessidade de conhecer muitos outros módulos.

Para ilustrar a coesão da abstração do modelo, as figuras 4.1 e 4.2 apresentam a estruturação do núcleo do OPENBUS utilizando o modelo original do SCS nas versões 1.4 e 1.5 respectivamente. Na figura 4.3, apresentamos a estruturação do sistema na versão 1.5 utilizando o modelo proposto. É possível notar que com o uso do modelo proposto mantém-se a identidade das facetas, pois elas representam um sub-serviço por inteiro, e não mais apenas uma versão do mesmo.

Essas figuras não incluem os componentes das aplicações que fazer uso do barramento ou que se conectam à ele. Perceba que, na atualização do sistema da versão 1.4 (figura 4.1) para a 1.5 (figura 4.2) no modelo antigo do SCS, os clientes que desejassem passar a utilizar a nova versão dos sub-serviços deveriam atualizar as conexões ou referências mantidas. Utilizando o modelo proposto (figura 4.3), os clientes que se enquadrassem nesse mesmo caso não precisariam atualizar as suas conexões e nem as referências mantidas. Apenas deveriam alterar a interface de acesso que utilizam para acessar o sub-serviço

PUC-Rio - Certificação Digital Nº 0912832/CA

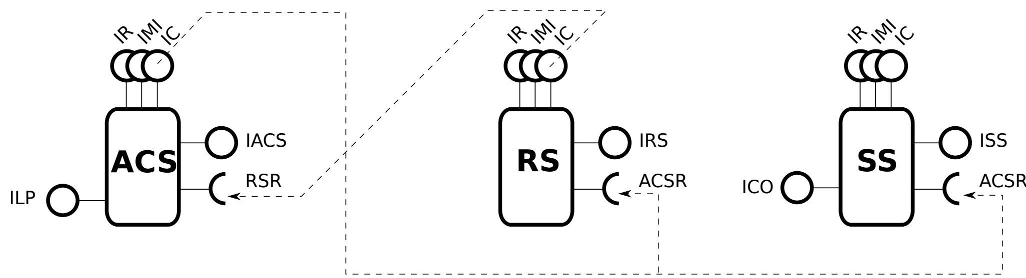

Figura 4.1: Arquitetura do OPENBUS na versão 1.4 utilizando o modelo original do SCS.

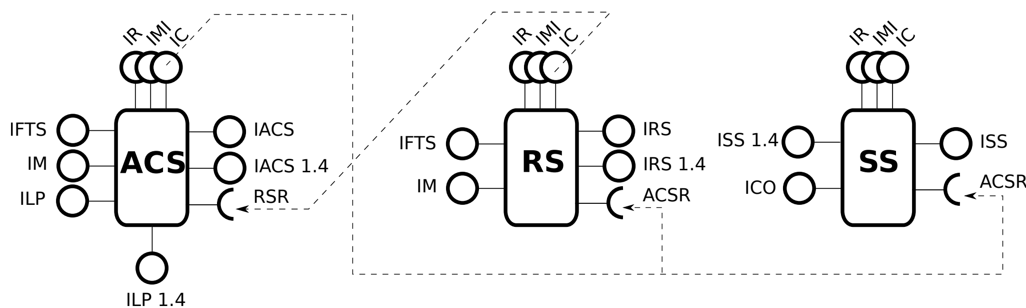

Figura 4.2: Arquitetura do OPENBUS na versão 1.5 utilizando o modelo original do SCS.

PUC-Rio - Certificação Digital Nº 0912832/CA

desejado. Essa é a maior contribuição que o nosso modelo oferece: facilitar o processo de evolução do sistema, mantendo uma visão mais abstrata de sua arquitetura.

Como forma de avaliar o modelo proposto, o nosso estudo de caso se mostrou bastante abrangente, pois com ele foi possível experimentar três dos quatro cenários de caso de uso mencionados na seção 3.3 e ilustrados na figura 3.3, que foram os cenários A (cliente e servidor sem MV), B (cliente sem MV e servidor com MV) e C (cliente e servidor com MV).

Um bom exemplo do relacionamento A é a versão 1.4 do OPENBUS, após a adequação para utilizar o modelo com suporte a múltiplas versões. Neste caso todos os relacionamentos existentes entre os componentes pertencem a este cenário. Em todos os casos, a versão que um cliente deseja era exatamente a única versão que o servidor oferece.

O cenário de uso do relacionamento B é um dos exemplos motivadores do modelo, pois ele ilustra a possibilidade do servidor evoluir sem obrigar a evolução simultânea dos clientes. Esse caso pode ser observado na realização dos demos e testes de uso do SDK da versão 1.4 do OPENBUS sobre a versão 1.5 com o suporte a múltiplas versões.

Um exemplo de relacionamento C pode ser visto no componente *Sessão*,

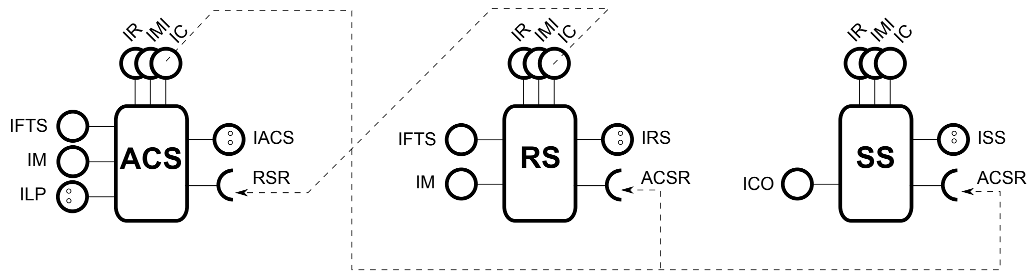

Figura 4.3: Arquitetura do OPENBUS na versão 1.5 utilizando o modelo do SCS com suporte a múltiplas versões.

que faz parte do mecanismo implementado no *Serviço de Sessão* na versão 1.5 do OPENBUS. Durante o cadastro de um novo membro na sessão, verifica-se a versão do sub-serviço de eventos oferecido pelo novo membro, e, encontrando uma das versões suportadas, o componente sessão realiza a comunicação de eventos através dessa versão compatível. O único tipo de relacionamento que não foi explorado explicitamente no estudo de caso foi o D (cliente com MV e servidor sem MV), mas ele é uma particularidade do relacionamento C, onde o conjunto de versões do lado do servidor tem apenas um elemento. Nesse cenário, a aplicação cliente deveria possuir uma implementação robusta, compatível com mais de uma versão, e assim poderia se conectar a qualquer componente com ou sem suporte a múltiplas versões, desde que o servidor disponibilizasse uma versão compatível com o cliente.

PUC-Rio - Certificação Digital Nº 0912832/CA