Especificação Hierárquica de Estilos Arquiteturais Distribuídos

Cidcley T. de Souza
Centro Federal de Educação Tecnológica do Ceará, Gerência de Telemática
NASH – Núcleo Avançado de Engenharia de Software Distribuído
e Sistemas Hipermídia
Fortaleza-CE, Brazil, 60040-531
cidcley@cefetce.br

Resumo

Estilos arquiteturais podem ser definidos com famílias de especificações arquiteturais obtidas a partir de generalizações de descrições arquiteturais específicas. Desse modo, a abordagem geral para a descrição de um estilo arquitetural é baseada na imposição de regras pré-estabelecidas às descrições arquiteturas seguindo o vocabulário e as restrições definidas para o estilo. Assim, a construção de um novo estilo requer a descrição de uma arquitetura que siga o estilo. Nesse artigo propomos uma abordagem inovadora para a descrição de estilos arquiteturais. Argumentamos aqui que estilos arquiteturais podem ser mais facilmente descritos se considerarmos uma abordagem hierárquica, onde a definição de novos estilos é baseada nas descrições de estilos já existentes. Para validar nossa abordagem apresentamos nesse artigo a linguagem Xtyle, que é uma aplicação de XML para a especificação de estilos arquiteturais distribuídos utilizando o conceito de herança múltipla. Além disso, apresentamos uma taxonomia para estilos arquiteturais distribuídos baseado em reuso de especificações.

Palavras chaves: Arquitetura de Software, Estilos Arquiteturais, Projeto Hierárquico.

Abstract

Architectural styles can be defined with families of architectural specifications obtained from generalizations of specific architectural descriptions. Therefore, the general approach to describe an architectural style is based on the imposition of pre-established rules to architectural descriptions following a vocabulary and the restrictions defined to the style. This way, the development of a new style requires de description of an architecture which follows the style. In this paper we propose an innovative approach for the description of architectural styles. We argue that architectural styles can be described more easily when considering a hierarchical approach, where the definition of new styles is based on the description of existing ones. In order to validate our approach we present the language Xtyle, which is an application of XML to the specification of distributed architectural styles using the concept of multiple inheritance. In addition, we present a taxonomy of distributed architectural styles based on reuse of specifications.

Keywords: Software Architecture, Architectural Styles, Hierarchical Design.

Abstract

Architectural styles can be defined with families of architectural specifications obtained from generalisations of specific architectural descriptions. This way, the general strategy of description of an architectural style is based on the imposition of pre-established rules to architectural descriptions following the vocabulary and the restrictions defined to the style. In this paper we propose a novel approach to the description of architectural styles. We argue that architectural styles can be described more easily if we consider a hierarchical approach, where the definitions of new styles is based on descriptions of pre-existing styles. In order to validate our approach we present the language Xtyle, an XML application to the specification of distributed architectural styles using the concept of multiple inheritance. In addition, we present a taxonomy to distributed architectural styles based on the reuse of specifications.

Keywords: Software Architecture, Architectural Style, Hierarchical Design.

1. INTRODUÇÃO

A indústria de software tem se defrontado atualmente com grandes desafios, dentre os quais, podemos destacar o aumento do tamanho e complexidade dos sistemas. Além da constante busca pela redução de esforços, custos, tempo de desenvolvimento e manutenção do software. Neste contexto, o reuso de software [14] é reconhecido como uma necessidade, principalmente nas fases iniciais do projeto, onde benefícios importantes podem ser alcançados, como por exemplo, a redução de tempo de desenvolvimento.

Diante deste cenário, a arquitetura de software [17, 20] tem se mostrado uma disciplina realmente útil por desempenhar um papel importante no processo de desenvolvimento do software, separando os elementos de computação (componentes) dos mecanismos de comunicação (conectores). Esta separação permite e facilita a promoção do reuso em níveis abstratos, que é um dos principais benefícios fornecidos pelo desenvolvimento arquitetural. Entretanto, para que os benefícios da arquitetura de software sejam realmente alcançados, ela deve ser tratada explicitamente servindo como base para análise, projeto e implementação. Atualmente as Linguagens de Descrição de Arquitetura (ADLs) são utilizadas para realizar essa tarefa.

O termo Estilos Arquiteturais de Software se refere a "um conjunto de regras de projeto que identifica os tipos de componentes e conectores que devem ser utilizados para compor um sistema ou subsistema, juntamente com um conjunto de restrições globais e locais na forma como essa composição deve ser realizada" [18].

Garlan em [11], apresenta de forma mais detalhada um conjunto de aspectos de estilos arquiteturais que devem ser utilizados por qualquer modelo que trabalhe com esse conceito. Ele apresenta os seguintes aspectos:

- **Vocabulário:** Deve-se ser adotado um conjunto de valores a serem definidos para os componentes de uma arquitetura de forma a permitir a verificação de compatibilidade com relação a um estilo, que também deve ser identificado, também por convenção, na descrição do sistema como um todo. Esses valores definem papéis que os componentes devem assumir com relação ao estilo requerido. Dessa forma, supondo que um sistema deva seguir o estilo arquitetural definido como **Pipeline**, os componentes que formam esses estilos devem possuir interfaces definidas com tipos como `Filter`, `Source` ou `Sink`. Assim, esses componentes assumem os papéis devidos com relação ao estilo.

- **Definição de Restrições Arquiteturais**: Além de definir as convenções sintáticas que permitem a verificação da compatibilidade dos tipos das interfaces dos componentes utilizados com o estilo arquitetural do sistema como um todo, também devemos criar um conjunto de regras topológicas de forma a garantir que esses componentes estejam respeitando as restrições definidas pelo estilo. Assim, um sistema que segue o estilo **Pipeline**, por exemplo, deve possuir um componente do tipo `Source` que esteja conectado com um componente do tipo `Filter` que, por sua vez deve ser conectado com um componente do tipo `Sink`. Além disso, a comunicação deve ser realizada sem ciclos e de forma unidirecional.

A idéia de estilos arquiteturais nas ADLs atuais ou é muito informal ou é demasiadamente formal. Em UniCon [19], por exemplo, existe um conjunto de restrições sintáticas que devem ser utilizados para a criação de arquiteturas que sigam determinados estilos. Contudo, essas restrições são baseadas em um conjunto de tipos de papéis que componentes podem assumir e esses tipos são fixos, tornando a linguagem restrita aos tipos disponíveis. Já em Wright [2], o desenvolvedor pode produzir seus próprios estilos mas para isso deve lançar mão de especificações algébricas baseadas em CSP o que não é um atrativo para os desenvolvedores.

Para facilitar a utilização de arquiteturas de software e estilos arquiteturais distribuídos, propusemos em [21] um *framework* de suporte a etapas de especificação, análise e implementação de arquiteturas de software distribuídas. Denominamos esse *framework* de DraX. Nesse trabalho, apresentamos uma linguagem, denominada Xtyle, apropriada para a especificação de estilos arquiteturais. A linguagem Xtyle será incorporada ao *framework* DraX e dará suporte a tarefa de especificação de estilos arquiteturais distribuídos

Nesse artigo apresentamos a linguagem Xtyle e mostramos a aplicação dessa linguagem na descrição de estilos arquiteturais distribuídos. Como principal contribuição desse trabalho, apresentamos uma forma nova de construir especificações de estilos arquiteturais. Baseado nas taxonomias para estilos arquiteturais propostas por Shaw [18], observamos que existe uma forma recorrente nas descrições desses estilos. Assim, muitos estilos compartilham elementos gramaticais e restrições estruturais. Desse modo, materializamos em Xtyle a idéia de produzir especificações

hierárquicas de estilos arquiteturais. Nessas especificações, podemos utilizar o conceito de herança múltipla para especificar estilos novos a partir de estilos pré-existentes. Isso permite ao desenvolvedor de especificações arquiteturais contar com um vocabulário irrestrito de estilos para especificar suas arquiteturas. Podendo, sempre que necessário, generalizar um estilo já existente de forma a melhor se edequar às suas necessidades.

Esse trabalho está estruturado da seguinte forma: nesta seção apresentamos as motivações desse trabalho e os principais conceitos de estilos arquiteturais. Na seção 2 apresentamos de forma simplificada o *framework* DraX. Na seção 3 apresentamos as idéias de especificação hierárquica de estilos arquiteturais fornecida em DraX. Na seção 4 apresentamos a linguagem Xtyle, onde os aspectos de descrição sintática de estilos arquiteturais são apresentados. Na seção 5 apresentamos um estudo de caso simples para mostrar a facilidade introduzida por Xtyle na extensão da especificação de um estilo arquitetural. Finalizamos na seção 6 com algumas conclusões e direcionamentos futuros.

2. O FRAMEWORK DRAX

Em [5] é demonstrado que as ADLs disponíveis atualmente não fornecem mecanismos satisfatórios para produzir especificações arquiteturais que possam ser facilmente implementadas sobre infraestruturas de *middleware*, visto que esses infraestruturas induzem diversas características que não são capturadas pelas ADLs. Ao mesmo tempo, temos que os desenvolvedores de software atualmente estão utilizando extensivamente infraestruturas de *middleware* como CORBA e RMI para facilitar a construção de aplicações distribuídas [6].

Baseado nessas idéias, desenvolvemos o *framework* DraX [21]. DraX (*DistRibuted Architecture based on XML*) é formado por um conjunto de linguagens, esquemas e *scritps* que podem ser utilizados em conjunto para permitir a descrição, validação, análise e implementação de arquiteturas de software e de estilos arquiteturais distribuídos sobre infraestruturas de *middleware*.

2.1 Metodologia de Desenvolvimento

DraX possui um conjunto de linguagens, esquemas e *scripts* que permitem realizar todas as fases do desenvolvimento arquitetural, desde a especificação das arquiteturas e estilos, passando pela validação sintática, estrutural e comportamental até a geração de *templates* de implementação. Desse modo, como o ferramental de DraX é relativamente extenso, resolvemos criar uma metodologia que possa guiar os passos de desenvolvimento de arquiteturas com o *framework*.

Nesse sentido, podemos visualizar duas etapas de desenvolvimento distintas e paralelas que interagem entre si. Na primeira etapa a arquitetura da aplicação é descrita; na segunda etapa o estilo arquitetural é descrito, caso esse ainda não tenha sido especificado. O resultado do desenvolvimento de um estilo é sua especificação em Xtyle devidamente validada. Essa especificação poderá ser referenciada na etapa de descrição de uma arquitetura ArchML. A tarefa de validação dessa arquitetura, entre outras coisas, verificará a aderência da mesma ao estilo referenciado.

É bom observar que a etapa de desenvolvimento de estilos não é comumente usada, pois DraX possui uma base de estilos pré-definidos que podem ser referenciados para criar arquiteturas, entretanto, é um diferencial de DraX com relação a outras ADLs e/ou ambientes de descrição de arquiteturas, o fato de o desenvolvedor contar com uma linguagem específica para definir seus próprios estilos arquiteturais. Além disso, esses novos estilos tanto podem ser definidos a partir de refinamentos dos estilos pré-existentes em DraX, como podem ser totalmente definidos.

De qualquer modo, a metodologia geral que propomos para a utilização de DraX para especificação de arquiteturas distribuídas é composta pelas seguintes etapas:

1. **Especificação**: Nessa fase são especificadas as arquiteturas e os estilos arquiteturais utilizando as linguagens ArchML e Xtyle respectivamente;
2. **Validação**: A verificação sintática, estrutural e comportamental das arquiteturas e estilos é realizada nessa fase. Sendo que são utilizados os esquemas e *scripts* de validação e verificação de DraX;
3. **Geração de Código**: Os *templates* de implementação são gerados a partir das especificações arquiteturais previamente validadas e das informações dos estilos que essas arquiteturas utilizam.

A Figura 1 apresenta o diagrama geral de atividades a serem realizadas para se descrever uma arquitetura em DraX. Como pode ser observado, existe um paralelismo entre as etapas de descrição de arquiteturas e de descrição de estilos, como foi explicado anteriormente.

É importante ressaltar nesse ponto que não estamos apresentando um processo de desenvolvimento de software distribuído baseado em arquitetura. Mesmo sabendo que apresentamos uma metodologia de aplicação das ferramentas de DraX para a produção de software distribuído baseado em arquitetura, não nos preocupamos em formalizar os passos e interações relativas a essas ferramentas. Dessa forma, por exemplo, aspectos como refinamento de especificações arquiteturais [9, 7] não são considerados nesse trabalho, mesmo sabendo que essa etapa pode ser realizada por ocasião da construção das especificações ou até mesmo diretamente nos códigos gerados a partir das ferramentas de DraX.

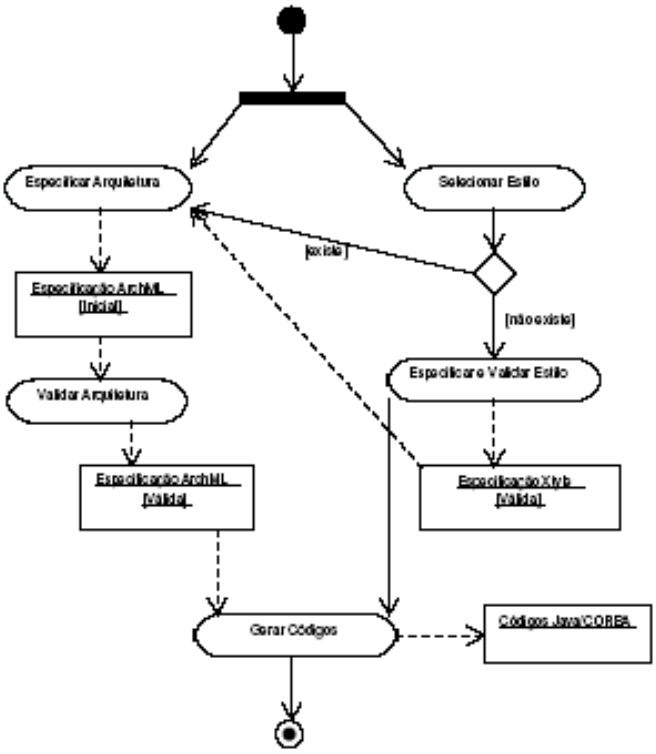

Figura 1. Metodologia de Desenvolvimento com DraX

3. HERANÇA NA DESCRIÇÃO DE ESTILOS ARQUITETURAIS

Em diversos trabalhos que apresentam algum tipo de taxonomia para estilos arquiteturais, como [19, 18, 12, 10, 17]. Pode ser facilmente observado o compartilhamento de informações entre diferentes estilos arquiteturais. Como por exemplo, alguns tipos de componentes estão presentes em diversos estilos diferentes. Assim como também algumas restrições topológicas. Dessa forma, nesse trabalho apresentamos uma nova abordagem de classificar os estilos de DraX tendo como parâmetro a existência de um vocabulário e de regras de configuração pré-definidos. Baseado nessa idéia, foram definidos três tipos de estilos: Estilos Básicos, Estilos Derivados e Estilos Definidos-Pelo-Usuário.

Os estilos básicos fornecem um conjunto de componentes descritos através de um vocabulário e comportamentos pré-definidos para cada componente participante desse estilo. Esses estilos, encontram um conjunto de scripts definidos em DraX para a validação de arquiteturas baseadas neles. Isso é possível pelo fato de esses serem estilos clássicos e de características relativamente bem definida.

Os estilos derivados são os que herdam as definições (vocabulários e comportamentos) de outros estilos definidos a priori. Esse tipo de estilo possui uma sub-classificação definida por: Derivados Simples e Derivados Compostos. Os estilos Derivados Simples herdam as características de apenas um estilo específico, alterando algumas de suas características. Já os estilos Derivados Compostos são os que herdam de mais de um estilo. Esses estilos também podem usar o ferramental disponível em DraX para realizar verificação em arquiteturas.

A seguir é apresentada uma taxonomia baseada no conceito de hierarquia de especificações para os estilos arquiteturais suportados por DraX é apresentada em detalhes. Esses estilos estão agrupados utilizando a classificação definida a pouco. Nessa seção serão apresentados os estilos, os tipos de componentes que podem fazer parte dele, a

descrição informal do comportamento de cada um e as regras de comunicação e restrições topológicas de cada um. Na próxima seção apresentaremos a linguagem Xtyle, na qual esses estilos são especificados.

Para especificar esses estilos, usamos os seguintes itens:

- **Descrição Geral**: Onde o estilo é apresentado de forma geral. Essa apresentação é acompanhada de uma ilustração para que se possa ter uma visão topológica de uma arquitetura que segue cada estilo.
- **Componentes:** Os componentes que formam o estilo são apresentados nesse momento. Assim, tanto os nomes que cada tipo de componente recebe como uma descrição geral de seu comportamento é apresentada. Essas informações servem de base para o vocabulário do estilo.
- **Restrições:** As restrições tanto relativas aos componentes (quantas portas pode ter, quantas conexões cada porta deve suportas, etc) como topológicas (quais tipos de componentes podem se comunicar e quais não podem, etc.) são levantadas nesse momento.
- **Fluxo de Dados e Controle:** Nesse item são apresentadas informações sobre o fluxo de dados (em que sentido(s) os dados caminham) e sobre o fluxo de controle (o dado é requisitado, os dados são enviados sem necessidade de pedidos, etc.).

3.1 Exemplo de Estilo

Para exemplificar a especificação informal realizada nesse trabalho de forma a se traças as relações de herança entre os estilos definidos em DraX, tomemos como exemplo o estilo simples denominado **Rede de Fluxo de Dados**.

- **Descrição Geral:** Na classificação de Shaw [18], **Rede de Fluxo de Dados** (*Dataflow Network*), ou DFN para simplificar, descreve estilos arquiteturais cujos componentes operam disponibilizando continuamente streams de dados e que são organizados em topologias arbitrárias. Abowd, Allen e Garlan [10] descrevem formalmente um estilo denominado Pipe-and-Filter, que na verdade corresponde as estilo DFN, e sugerem a definição de um estilo que é formado por componentes que transportam dados de entrada em dados de saída de forma assíncrona e com mínima retenção e possuem uma topologia arbitrária.
- **Componentes:** Os tipos de componentes definidos para este tipo de estilo são os Filtros (`Filters`), que recebem dados nas suas portas de entrada e os mandam pelas portas de saída. Contudo existe uma especialização desse tipo de componente que é bastante aceita na literatura [16, 15, 3, 6] e são denominados de `Sources` e `Sinks`. Assim, `Sources` identificam as fontes de dados e `Sinks` são os consumidores finais dos dados.
- **Restrições:** Dois tipos de restrições são definidos para esse estilo. Inicialmente as restrições relativas às portas dos componentes. Em [18] é definido que os componentes do tipo `Filter` devem ter pelo menos uma porta de entrada e pelo menos uma porta de saída. Em [16, 15] é indicado que os componentes do tipo `Source` devem ter exatamente uma porta de entrada e pelo menos uma porta de saída e os componentes do tipo `Sink` devem ter exatamente uma porta de saída e pelo menos uma porta de entrada.
- O outro tipo de restrição é a topológica. Pela definição usual [12, 18, 2], esses estilos devem apenas garantir que os dados que entram nos componentes do tipo `Filter` sejam enviados pelas portas de saída desses. Assim, a topologia é arbitrária [1, 18]. Obviamente que existem diversas variações [18, 16], entretanto aqui o que interessa é a definição mais geral para o estilo, já que as variações devem ser criadas a partir de refinamentos [8], sendo que algumas dessas variações são suportadas por DraX e serão apresentadas como estilos derivados mais adiante.
- **Fluxo de Dados e Controle:** O fluxo de dados entre os componentes, seguindo a taxonomia de Shaw [18] é realizado através da passagem de streams de um componente do tipo `Filter` para outro. Portanto não existem dados compartilhados entre os Filtros. Já o fluxo de controle, também definido em Shaw [18] é considerado o mesmo dos dados, ou seja, quem gera os dados é quem os passa adiante.

4. A LINGUAGEM XTYLE

Até agora a descrição de estilos arquiteturais está restrita a informalização da prosa. Contudo, podemos observar que na descrição das arquiteturas em ArchML definimos uma referência a um arquivo externo onde o estilo está definido para que se possam ser realizadas as verificações e validações devidas com relação a aderência de uma arquitetura a um estilo. Portanto, esse estilo deve ser definido nesse arquivo.

Uma característica marcante de DraX é possuir uma linguagem específica para de descrever estilos arquiteturais. Essa linguagem, denominada Xtyle, permite se descrever tanto o vocabulário permitido pelo estilo, como as restrições de comunicação e topológicas. Atualmente as abordagens para se descrever estilos se polarizam em

descrições informais e sujeitas a todas as imprecisões dos textos escritos [12] às descrições formais e difíceis de serem de fato implementadas. Contudo as duas visões são importantes [2]. No informalismo somos capazes de entender o funcionamento de um estilo, no formalismo podemos avaliar uma arquitetura que segue o estilo e examinar diversas propriedades que seriam impossíveis, ou no mínimo, muito difíceis de avaliar sem o uso de modelos matemáticos.

Baseado nesse pensamento, resolvemos que Xtyle permitiria tanto a informalização na descrição de vocabulários e regras de comunicação como o formalismo de álgebras de processos para a verificação de arquiteturas que usam um estilo. Assim, seguindo a mesma estrutura de ArchML, em Xtyle, usamos XML para descreve as estruturas sintáticas de um estilo e as restrições topologias e de comunicação.

A estrutura geral de uma especificação Xtyle é a apresentada na especificação a seguir. Podemos observar os elementos básicos de uma especificação de estilos se referem a: documentação, tipos e topologia.

```
<?xml version="1.0"?>
<xtyle name="nmtoken">
      <document/>
      <uses/>?
      <types/>
      <topology/>
</xtyle>
```

O elemento `xtyle`, que é a raiz da especificação, possui um atributo `name`, que serve para identificar o estilo que está sendo descrito. O elemento `document`, que também é usado em ArchML, serve para definir informações sobre a realização da especificação. O elemento `types`, permite a identificação dos tipos de componentes permitidos para o estilo e o elemento `topology`, permite se definir aspectos topológicos.

O elemento `document` é utilizado para se descrever textualmente a funcionalidade do estilo. Utilizado para fins documentais, esse elemento não é processado em nenhuma etapa e serve, apenas, para controle por parte da equipe de desenvolvimento.

Quando descrevermos um estilo derivado, temos que definir de qual(is) estilo(s) ele deriva. O elemento `uses` serve para esse fim. A sintaxe desse elemento é a seguinte:

```
<uses>
      <style name="Pipeline" href="Pipeline.xty"/>
</uses>
```

O elementos `uses` é formado por vários elementos do tipo `style`, que identifica um estilo dos quais estamos herdado a especificação. Cada elemento `style`, possui um `name`, que identifica o nome que será usado na especificação para referenciar o estilo herdado e o atributo `href` pelo qual podemos definir qual arquivo o estilo que estamos usando está descrito. O vocabulário do estilo é definido pelo elemento `types`. Além das informações sobre possíveis restrições nos componentes que utilizam esses tipos. O código a seguir apresenta um exemplo de especificação desse elemento.

```
<types>
  <type name="Filter" from="DataFlow">
  <alias name="Supplier" from="Pipeline"/>
       <ports>
         <in maxOccurs="1" maxConns="1"/>
         <out maxOccurs="1"/>
       </ports>
  </type>
  ...
</types>
```

O elemento `types` é formado por um conjunto de elementos `type`. Cada `type` identifica um tipo diferente de componente permitido o estilo. O atributo `name` de `type`, identifica o nome que o tipo recebe. Podemos observa um atributo `from`. Esse atributo é usado na descrição de estilos derivados para indicar de qual estilo estamos herdando o tipo, visto que um estilo pode herdar de diversos outros. Além disso, o elemento filho `alias` permite se definir as

correlações entre tipos em um estilo derivado. Esse elemento será melhor entendido quando apresentarmos a aplicação de Xtyle na descrição dos estilos de DraX.

Cada elemento `type`, também possui como elementos filhos, os elementos `in` e `out`. O elemento in, define que o tipo deve possuir portas de entrada, sendo que os atributos desse elemento indicam as restrições relativas a essas portas. O mesmo ocorre para o elemento out, que descrever portas de saída.

Os elementos `in` e `out` possuem diversos atributos opcionais. O atributo `minOccurs`, define a quantidade mínima de portas de entrada/saída os componentes que seguem esse tipo devem possuir, sendo que o valor default é 1 e pode também ser definido o valor "*" para um número ilimitado de portas. O mesmo ocorre para o atributo `maxOccurs`, que define o número máximo de portas de entrada/saída que o componente que segue esse tipo deve possuir. O atributo `minConns` e `maxConns` identificam, respectivamente, o número mínimo é máximo de conexões que uma porta de entrada/saída deve permitir.

O elemento `topology` define informações sobre as interações entre os tipos definidos para o estilo. Um exemplo desse elemento é apresentado a seguir.

```
<topology>
  <link name="SupplierConsumer" start="Supplier"
        end="Consumer" type="push"/>
</topology>
```

O elemento `topology` é formado por um conjunto de elementos do tipo `link`. Cada elemento `link`, representa uma conexão permitida entre os tipos definidos no estilo. O atributo `name`, define um nome para a conexão e o atributo `from`, que é opcional, é utilizado na descrição de estilos derivados e apresenta o nome do estilo do qual são herdadas essas descrições de conexão. Os atributos `start` e `end` de `link`, definem os nomes dos tipos que podem ser conectados. O atributo opcional `controlType` identifica o tipo de comunicação entre esses componentes, tendo os valores possíveis `push` ou `pull`, tendo `push` como default.

4.1 Exemplo

Para exemplificar a aplicação de Xtyle na descrição de um estilo suportado por DraX usando Xtyle apresentamos aqui a especificação do estilo **Rede de Fluxo de Dados**. Em [22] podem ser encontradas as descrições completas de todos os demais estilos suportados por DraX. Essa especificação é baseada na especificação informal apresentada na seção anterior. A especificação Xtyle de **Rede de Fluxo de Dados** é mostrada no código abaixo.

```
<?xml version="1.0"?>
<xtyle name="DataFlowNetwork">
  <document>
  </document>
  <types>
       <type name="Filter">
         <ports>
              <in mode="async"/>
              <out mode="async"/>
         </ports>
         <behavior href="Filter.xmi"/>
       </type>
       <type name="Source">
         <ports>
              <out mode="async"/>
         </ports>
         <behavior href="Source.xmi"/>
       </type>
       <type name="Sink">
         <ports>
              <in mode="async"/>
         </ports>
         <behavior href="Sink.xmi"/>
       </type>
  <topology>
       <link name="SourceFilter" start="Source" end="Filter"
             type="push"/>
       <link name="FilterSink" start="Filter" end="Sink"
             type="push"/>
       <link name="FilterFilter" start="Filter" end="Filter"
             type="push"/>
  </topology>
</xtyle>
```

5. ESTUDO DE CASO

Para um melhor entendimento da abordagem que estamos propondo nesse artigo para a especificação de estilos arquiteturais baseadas em herança entre especificações de estilos a partir das definições de outros estilos pré-existentes, tomemos como exemplo o estilo arquitetural denominado **Pipeline Event**. Esse estilo é um estilo derivado composto de DraX. A derivação desse estilo é feita a partir dos estilos **Pipeline** e **Event**. Sendo que o estilo **Pipeline** já é um estilo derivado de **Rede de Fluxo de Dados**, cujas especificações foram apresentadas nos exemplos das seções anteriores. Segue abaixo as descrições informais e Xtyle de todos esses estilos.

5.1 Estilo Pipeline

Especificação Informal

- **Descrição Geral:** Esse estilo é uma derivação de Rede de Fluxo de Dados onde algumas restrições topológicas são incluídas.
- **Estilos Base**: Esse estilo é derivado de Rede de Fluxo de Dados.
- **Componentes:** Os tipos de componentes desse estilo são os mesmos apresentados em Rede de Fluxo de Dados.
- **Restrições:** As restrições impostas tanto aos componentes quanto à topologia de Rede de Fluxo de Dados é que caracterizam esse estilo. Assim, um Pipeline não deve possuir ciclos e deve ter uma topologia linear. Para garantir essa propriedade impomos a restrição de que cada componente do tipo `Filter` só pode possuir uma única porta de entrada e uma única porta de saída, sendo que o número máximo de conexões nessas portas é 1. Já os componentes do tipo `Source` devem ter exatamente uma porta de saída e os componentes do tipo `Sink` devem ter exatamente uma porta de entrada. Além disso a comunicação ocorre de forma síncrona.
- **Fluxo de Dados e Controle:** Tanto o fluxo de dados como de controle nesse estilo são idênticos aos apresentados em Rede de Fluxo de Dados.

Especificação Xtyle

```
<?xml version="1.0"?>
<xtyle name="Pipeline">
  <document>
  </document>
  <uses>
       <style name="DataFlow" href="DataFlowNetwork.xty/>
  </uses>
  <types>
       <type name="Filter" from="DataFlow">
         <ports>
              <in maxOccurs="1" maxConns="1"/>
              <out maxOccurs="1"/>
         </ports>
       </type>
       <type name="Source" from="DataFlow">
         <ports>
              <in maxOccurs="1"/>
         </ports>
       </type>
       <type name="Sink" from="DataFlow">
         <ports>
              <out maxOccurs="1"/>
         </ports>
       </type>
  </types>
  <topology>
       <link name="SourceFilter" from="DataFlow"/>
       <link name="SinkFilter" from="DataFlow"/>
       <link name="FilterFilter" from="DataFlow"/>
  </topology>
</xtyle>
```

5.2. Estilo Event

Especificação Informal

- **Descrição Geral:** O estilo `Event`, é definido em [4] como um conjunto de componentes que se comunicam através da geração e recebimento de notificações de eventos, sendo que um componente gera uma notificação de evento quando tem a necessidade de expressar para o ambiente externo a ocorrência de um evento. Quando uma notificação de evento é gerada ela é propagada para todos os componentes que demonstraram interesse em receber notificação desse determinado evento.

- **Componentes:** Em [1] é definido que os componentes do estilo **Event** são definidos apenas como Objetos. Cada objeto possui um conjunto de métodos (portas) que podem ser invocados e transformar o estado interno do objeto gerando eventos. Um vocabulário bastante usado para definir os tipos de Objetos nesse estilo é o de Produtor (`Supplier`) [13] que define um objeto que gera notificação de eventos e Consumidor (`Consumer`), que se apresenta como interessado em determinados eventos. A propagação de notificações de eventos é realizada de fato por um Canal de Eventos (`Event Channel`) [13], Distribuidores (`Distributors`) [1] ou Serviço de Eventos (Event Service) [4]. Assim, os Produtores e Consumidores que compartilharem o mesmo Canal de Eventos, estarão aptos a trocarem eventos.

- **Restrições:** Cada Produtor deve gerar notificações de eventos apenas para os Consumidores que estiverem interessados no evento [4], sendo que essa notificação deve ocorrer de forma assíncrona. Essa restrição deve ser controlada pelo Canal de Eventos que deve repassar as notificações para os componentes corretos. Os Produtores devem possuir pelo menos uma porta de saída para enviar as notificações e os consumidores devem possuir pelo menos uma porta de entrada para receber as notificações de eventos.

- **Fluxo de Dados e Controle:** A forma tradicional de fluxo de dados apresentada para esse estilo ocorre no sentido do Produtor para o Consumidor [4]. Esse mesmo sentido é seguido para o fluxo de controle, onde o Produtor, depois de gerar a notificação de eventos, avisa os Consumidores que ela está disponível. É freqüentemente utilizado na literatura [4, 13] o nome de "push" para esse modelo de transmissão de dados do Produtor para o Consumidor.

Especificação Xtyle

```
<?xml version="1.0"?>
<xtyle name="Event">
  <document>
  </document>
  <types>
       <type name="Consumer">
         <ports>
              <in mode="async"/>
              <out minOccurs="0" mode="async"/>
         </ports>
       <behavior href="Consumer.xmi"/>
       </type>
       <type name="Supplier»
         <ports>
              <in minOccurs="0" mode="async"/>
              <out mode="async"/>
         </ports>
         <behavior href="Supplier.xmi"/>
       </type>
  <topology>
       <link name="SupplierConsumer" start="Supplier"
            end="Consumer" type="push"/>
  </topology>
</xtyle>
```

5.3. Estilo Pipeline Event

Especificação Informal

- **Descrição Geral:** Esse estilo, proposto em [24] é formado através da fusão do estilo Pipeline e Eventos onde os elementos trocam informações através de eventos e não de forma síncrona como é definido pelo estilo Pipeline.

- **Estilos Base**: Os estilos usados nesse estilo são os estilos Pipeline e Eventos.

- **Componentes:** Os componentes são os mesmos propostos por Pipeline, sendo que os tipos `Source` e `Sink` funcionam, respectivamente como Produtores e Consumidores de eventos e os Filtros são Consumidores de notificações de eventos que eles recebem de outros Filtros ou de `Source` e Produtores de notificações de eventos para outros Filtros ou para `Sink`.

- **Restrições:** O tipo de topologia é o mesmo de Pipeline e a comunicação ocorre como em Eventos e segue um esquema de armazena-envia-adiante *(store-forward*).

- **Fluxo de Dados e Controle:** O fluxo de dados se dá da mesma forma que em Pipeline, e o controle usa a idéia de *push* do estilo Eventos.

Especificação Xtyle

```
<?xml version="1.0"?>
<xtyle name="PipelineEvent">
  <document>
  </document>
  <uses>
       <style name="Pipeline" href="Pipeline.xty"/>
       <style name="Event" href="Event.xty"/>
  </uses>
  <types>
       <type name="Filter" from="Pipeline">
         <alias name="Supplier" from="Pipeline"/>
         <alias name="Consumer" from="Pipeline"/>
         <ports>
              <in mode="async"/>
              <out mode="async"/>
         </ports>
       </type>
       <type name="Source" from="Pipeline">
         <alias name="Supplier" from="Pipeline"/>
         <ports>
              <in mode="async"/>
         </ports>
       </type>
       <type name="Sink" from="Pipeline">
         <alias name="Consumer" from="Pipeline"/>
         <ports>
              <out mode="async"/>
         </ports>
       </type>
  </types>
  <topology>
       <link name="SourceFilter" from="Pipeline"/>
       <link name="SinkFilter" from="Pipeline"/>
       <link name="FilterFilter" from="Pipeline"/>
  </topology>
</xtyle>
```

6. CONCLUSÃO

Nesse artigo apresentamos uma nova abordagem para a definição de estilos arquiteturais. Baseado na observação de que existem diversas informações que são compartilhadas pelos mais diversos estilos arquiteturais, apresentamos uma taxonomia baseada na idéia de herança entre especificações. Na qual um conjunto de estilos básicos podem ser estendidos para a criação de estilos arquiteturais mais complexos.

Também apresentamos nesse trabalho a linguagem Xtyle. Essa linguagem, situada no contexto do *framework* DraX, tem a intenção de fornecer um mecanismo de fácil manipulação para a descrição de estilos arquiteturais em DraX. Dentre as diversas características novas introduzidas por Xtyle, podemos citar como a mais importante a possibilidade de criação de novos estilos a partir da herança de um ou mais estilos pré-definidos. Esses estilos podem ser criados através da restrição e extensão de outros estilos de forma rápida e simples.

Em Xtyle, além da definição do vocabulário de um estilo, ou seja, dos nomes dos papéis que os componentes devem assumir em uma arquitetura de modo a seguir um determinado estilo, também definimos outras restrições importantes, como às relativas ao número de portas de entrada e saída, ao tipo de comunicação realizada nessa porta (síncrona ou assíncrona). Além disso, também podemos definir restrições topológicas, que dizem respeito às interações entre os tipos de um estilo, refletindo informações relativas ao fluxo de dados entre os participantes do estilo e o fluxo de controle, ou seja, como a comunicação é controlada entre esses participantes.

Os estilos Básicos são os estilos que já possuem todas as ferramentas de manipulação disponíveis em DraX e que formam a base inicial de herança para os demais estilos. Os estilos Derivados são os que são formados se herdando definições de estilos de DraX e realizando adaptações nessas especificações. Um fato importante a ressaltar é que esses estilos podem ser derivados de um ou mais estilos. Em [23, 21] pode ser encontrada uma taxonomia completa dos estilos utilizados em DraX, com suas respectivas derivações.

Além disso, a linguagem Xtyle facilita o entendimento sobre o processo de descrição de estilos, normalmente realizado do modo formal em outras ADL. Dessa forma, Xtyle pode ser aplicado como um facilitador no ensino dos conceitos de estilos arquiteturais em disciplinas de Engenharia de Software.

Referências

[1] Abowd, G. and Allen, R. and Garlan, D. Formalizing Styles to Understand Descriptions of Software Architecture. *ACM Transactions on Software Engineering and Methodology*, 1995.

[2] Allen, R. and Garlan, D. A Formal Basis for Architectural Connections. IEEE Transactions on Software Engineering, 1997.

[3] Bass, L. and Clements, P. and Kazman, R. Software Architecture in Practice. Addison-Wesley, 1998.

[4] Carzaniza, A. and Di Nitto, E. and Resenblum, D. and Wolf, A. Issues in Supporting Event-Based Architectural Styles. In Proc. Third International Software Architecture Workshop, 1998.

[5] Di Nitto, Elisabetta and Rosenblum, David S. Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures. In International Conference on Software Engineering, pages 13–22, 1999.

[6] Di Nitto, Elisabetta and Rosenblum, David S. On the Role of Style in Selecting Middleware and Underwear. In Engineering Distributed Object 99, ICSE 9 workshop., 1999.

[7] Egyed, A. and Grunbacher, P. and Medvidovic, N. Refinement and Evolution Issues in Bridging Requirements and Architecture: The CBSP Approach. In Proc. of the From Software Requirements to Architectures Workshop (STRAW 2001), 2001.

[8] Garlan, D. What is a Style? In Proc. Dagshtul Workshop on Software Architecture, 1995.

[9] Garlan, D. Style-Based Refinement for Software Architecture. In Joint Proceedings of the Second International Software Architecture Workshop (ISAW2) and the International Workshop on Multiple Perspectives in Software Development(Viewpoints ’96). ACM Press, 1996.

[10] Garlan, D. and Allen, R. and Ockerbloom, J. Exploiting Style in Architectural Design Environments. In Proc. ACM SIGSOFT ’94 Symposium on Foundations of Software Engineering, 1994.

[11] Garlan, D. and Kompanek. A. and Melton, R. and Monroe, R. Architectural Style: An Object-Oriented Approach. Technical report, Carnegie Mellon University, 1996.

[12] Garlan, D. and Shaw, M. An Introduction to Software Architecture. In Advances in Software Engineering and Knowledge Engineering, I, 1993.

[13] Object Management Group. CORBA Services: Common Object Services Specification. In OMG Technical Document formal/97 -07-04, 1997.

[14] Jacobson, I. and Griss, M. and Jonsson, P. Software Reuse - Architecture, Process and Organization for Business Success. Addison-Wesley, 1997.

[15] Lea, D. Concurrent Programming in Java. Addison-Wesley, 1996.

[16] Manolescu, D. A Data Flow Pattern Language. In Proc. Patterns Languages of Programming (PLoP'97), 1997.

[17] Perry, D. and Alexander, L. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, 17(4), October 1992.

[18] Shaw, M. and Clements, P. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems, 1996.

[19] Shaw, M. and DeLine, R. and Klein, D.V. Abstractions for Software Architecture and Tools for Support Them. In IEEE Trans on Software Engineering, 1995.

[20] Shaw, M. and Garlan, D. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.

[21] Souza, Cidcley T. de. Arquiteturas de Software e Estilos Arquiteturais Distribuídos - Especificação, Validação, Análise e Implementação. Tese de Doutorado. Universidade Federal de Pernambuco, 2003.

[22] Souza, Cidcley T. de and Cunha, Paulo. R. F. Descrição de Arquiteturas e Estilos em DraX. Relatório Técnico. Universidade Federal de Pernambuco, 2003.

[23] Souza, Cidcley T. de and Cunha, Paulo. R. F. Especificação de Estilos em Arquiteturas de Software. In XXIX Conferência Latino-Americana de Informática., La Paz, Bolivia, 2003.

[24] Thakkar, A. and Ramamurthy, B. Event Pipeline Pattern. Technical report, State University of New York at Buffalo, 2001.