CAPÍTULO 5

Transferência Representacional do Estado (REST)

Este capítulo apresenta e elabora o estilo de arquitetura Representational State Transfer (REST) para sistemas de hipermídia distribuídos, descrevendo os princípios de engenharia de software que orientam o REST e as restrições de interação escolhidas para reter esses princípios, enquanto os contrasta com as restrições de outros estilos arquiteturais. REST é um estilo híbrido derivado de vários estilos de arquitetura baseados em rede descritos no Capítulo 3 e combinado com restrições adicionais que definem uma interface de conector uniforme. A estrutura de arquitetura de software do Capítulo 1 é usada para definir os elementos arquitetônicos do REST e examinar amostras de processo, conector e visualizações de dados de arquiteturas prototípicas.

5.1 Derivando REST

A lógica de design por trás da arquitetura da Web pode ser descrita por um estilo de arquitetura que consiste no conjunto de restrições aplicadas aos elementos da arquitetura. Ao examinar o impacto de cada restrição à medida que ela é adicionada ao estilo em evolução, podemos identificar as propriedades induzidas pelas restrições da Web. Restrições adicionais podem ser aplicadas para formar um novo estilo de arquitetura que reflita melhor as propriedades desejadas de uma arquitetura moderna da Web. Esta seção fornece uma visão geral do REST, percorrendo o processo de derivá-lo como um estilo arquitetônico. As seções posteriores descreverão com mais detalhes as restrições específicas que compõem o estilo REST.

5.1.1 Começando com o Estilo Nulo

Existem duas perspectivas comuns sobre o processo de projeto arquitetônico, seja para edifícios ou para software. A primeira é que um designer começa com nada – uma lousa em branco, quadro branco ou prancheta – e constrói uma arquitetura a partir de componentes familiares até que ela satisfaça as necessidades do sistema pretendido. A segunda é que um projetista começa com as necessidades do sistema como um todo, sem restrições, e então, de forma incremental, identifica e aplica restrições aos elementos do sistema para diferenciar o espaço de projeto e permitir que as forças que influenciam o comportamento do sistema fluam naturalmente, em harmonia com o sistema. Onde o primeiro enfatiza a criatividade e a visão ilimitada, o segundo enfatiza a contenção e a compreensão do contexto do sistema. REST foi desenvolvido usando o último processo.

O estilo Null (Figura 5-1) é simplesmente um conjunto vazio de restrições. De uma perspectiva arquitetural, o estilo nulo descreve um sistema no qual não há limites distintos entre os componentes. É o ponto de partida para nossa descrição de REST.

Figure 5-1. Null Style

5.1.2 Cliente-Servidor

As primeiras restrições adicionadas ao nosso estilo híbrido são aquelas do estilo arquitetural cliente-servidor (Figura 5-2), descritas na Seção 3.4.1 . A separação de interesses é o princípio por trás das restrições cliente-servidor. Ao separar as preocupações da interface do usuário das preocupações com o armazenamento de dados, melhoramos a portabilidade da interface do usuário em várias plataformas e melhoramos a escalabilidade simplificando os componentes do servidor. Talvez o mais significativo para a Web, no entanto,

seja que a separação permite que os componentes evoluam independentemente, suportando assim o requisito de escala da Internet de vários domínios organizacionais.

Figure 5-2. Client-Server

5.1.3 Apátrida

Em seguida, adicionamos uma restrição à interação cliente-servidor: a comunicação deve ser sem estado por natureza, como no estilo cliente-servidor-sem estado (CSS) da Seção 3.4.3 (Figura 5-3), de modo que cada solicitação do cliente para O servidor deve conter todas as informações necessárias para entender a solicitação e não pode tirar proveito de nenhum contexto armazenado no servidor. O estado da sessão é, portanto, mantido inteiramente no cliente.

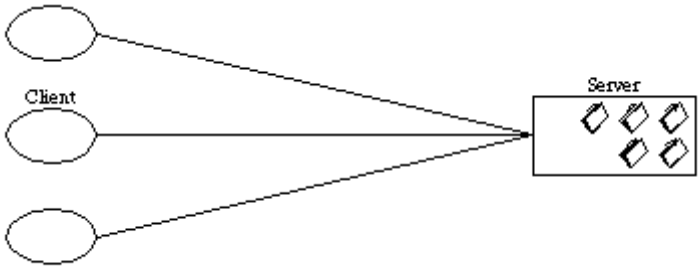

Figure 5-3. Client-Stateless-Server

Essa restrição induz as propriedades de visibilidade, confiabilidade e escalabilidade. A visibilidade é melhorada porque um sistema de monitoramento não precisa olhar além de um único dado de solicitação para determinar a natureza completa da solicitação. A confiabilidade é melhorada porque facilita a tarefa de recuperação de falhas parciais [133]. A escalabilidade é aprimorada porque não ter que armazenar o estado entre as solicitações permite que o componente do servidor libere recursos rapidamente e simplifica ainda mais a implementação porque o servidor não precisa gerenciar o uso de recursos entre as solicitações.

Como a maioria das escolhas arquitetônicas, a restrição stateless reflete uma troca de design. A desvantagem é que pode diminuir o desempenho da rede aumentando os dados repetitivos (sobrecarga por interação) enviados em uma série de solicitações, pois esses dados não podem ser deixados no servidor em um contexto compartilhado. Além disso, colocar o estado do aplicativo no lado do cliente reduz o controle do servidor sobre o comportamento consistente do aplicativo, pois o aplicativo se torna dependente da implementação correta da semântica em várias versões do cliente.

5.1.4 Cache

Para melhorar a eficiência da rede, adicionamos restrições de cache para formar o estilo client-cache-stateless-server da Seção 3.4.4 (Figura 5-4). As restrições de cache exigem que os dados em uma resposta a uma solicitação sejam rotulados implícita ou explicitamente como armazenáveis em cache ou não armazenáveis em cache. Se uma resposta puder ser armazenada em cache, um cache de cliente terá o direito de reutilizar esses dados de resposta para solicitações equivalentes posteriores.

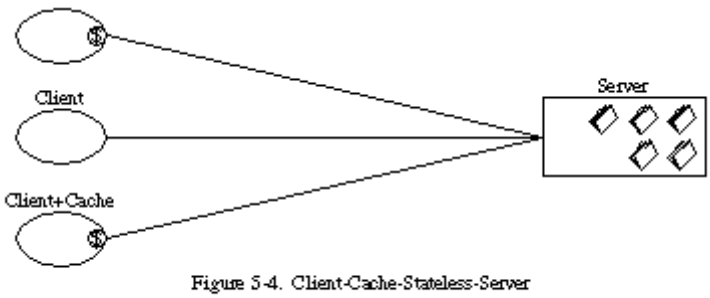

Figure 5-4. Client-Cache-Stateless-Server

A vantagem de adicionar restrições de cache é que elas têm o potencial de eliminar parcial ou completamente algumas interações, melhorando a eficiência, a escalabilidade e o desempenho percebido pelo usuário, reduzindo a latência média de uma série de interações. A desvantagem, no entanto, é que um cache pode diminuir a confiabilidade se os dados obsoletos dentro do cache diferirem significativamente dos dados que seriam obtidos se a solicitação tivesse sido enviada diretamente ao servidor.

A arquitetura da Web inicial, conforme retratado pelo diagrama na Figura 5-5 [11], foi definida pelo conjunto de restrições cliente-cache-stateless-server. Ou seja, a lógica de design apresentada para a arquitetura da Web antes de 1994 focava na interação cliente-servidor sem estado para a troca de documentos estáticos pela Internet. Os protocolos para comunicação de interações tinham suporte rudimentar para caches não compartilhados, mas não restringiam a interface a um conjunto consistente de semânticas para todos os recursos. Em vez disso, a Web dependia do uso de uma biblioteca de implementação cliente-servidor comum (CERN libwww) para manter a consistência entre os aplicativos da Web.

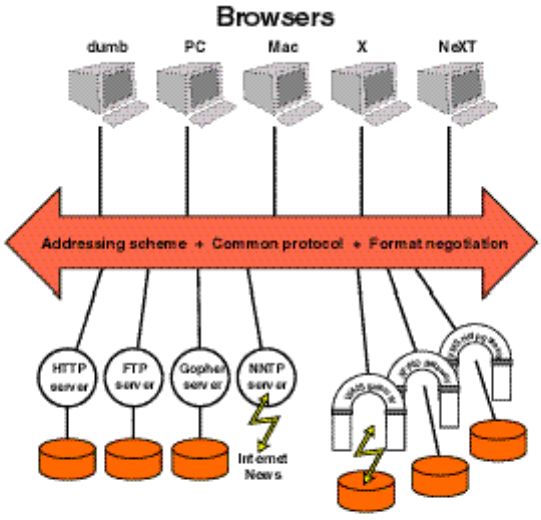

Figure 5-5. Early WWW Architecture Diagram

Os desenvolvedores de implementações da Web já haviam ultrapassado o design inicial. Além de documentos estáticos, as solicitações podem identificar serviços que geram respostas dinamicamente, como mapas de imagem [Kevin Hughes] e scripts do lado do servidor [Rob McCool]. O trabalho também havia começado em componentes intermediários, na forma de proxies [79] e caches compartilhados [59], mas eram necessárias extensões aos protocolos para que eles se comunicassem de forma confiável. As seções a seguir descrevem as restrições adicionadas ao estilo de arquitetura da Web para orientar as extensões que formam a arquitetura moderna da Web.

5.1.5 Interface Uniforme

A característica central que distingue o estilo de arquitetura REST de outros estilos baseados em rede é sua ênfase em uma interface uniforme entre os componentes (Figura 5-6). Ao aplicar o princípio de generalidade da engenharia de software à interface do componente, a arquitetura geral do sistema é simplificada e a visibilidade das interações é aprimorada. As implementações são dissociadas dos serviços que fornecem, o que incentiva a evolução independente. A desvantagem, porém, é que uma interface uniforme degrada a eficiência, uma vez que as informações são transferidas de uma forma padronizada e não de uma forma específica para as necessidades de um aplicativo. A interface REST foi projetada para ser eficiente para transferência de dados hipermídia de grande granularidade, otimizando para o caso comum da Web, mas resultando em uma interface que não é ideal para outras formas de interação arquitetônica.

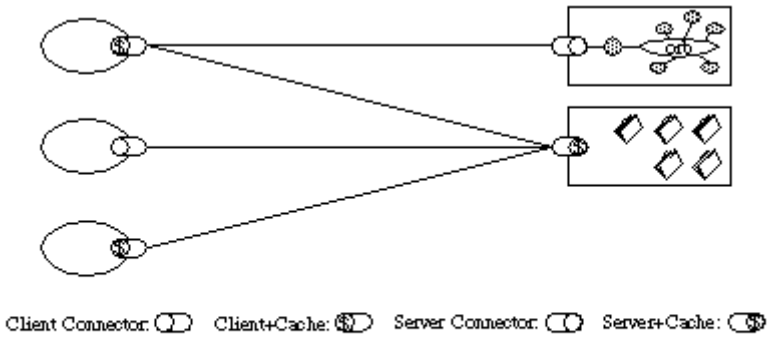

Figure 5-6. Uniform-Client-Cache-Stateless-Server

Para obter uma interface uniforme, são necessárias várias restrições arquiteturais para guiar o comportamento dos componentes. REST é definido por quatro restrições de interface: identificação de recursos; manipulação de recursos por meio de representações; mensagens autodescritivas; e, hipermídia como motor do estado da aplicação. Essas restrições serão discutidas na Seção 5.2 .

5.1.6 Sistema em camadas

Para melhorar ainda mais o comportamento dos requisitos de escala da Internet, adicionamos restrições de sistema em camadas (Figura 5-7). Conforme descrito na Seção 3.4.2, o estilo de sistema em camadas permite que uma arquitetura seja composta de camadas hierárquicas restringindo o comportamento do componente de modo que cada componente não possa "ver" além da camada imediata com a qual está interagindo. Ao restringir o conhecimento do sistema a uma única camada, limitamos a complexidade geral do sistema e promovemos a independência do substrato. As camadas podem ser usadas para encapsular serviços herdados e proteger novos serviços de clientes herdados, simplificando componentes ao mover funcionalidades raramente usadas para um intermediário compartilhado. Os intermediários também podem ser usados para melhorar a escalabilidade do sistema, permitindo o balanceamento de carga de serviços em várias redes e processadores.

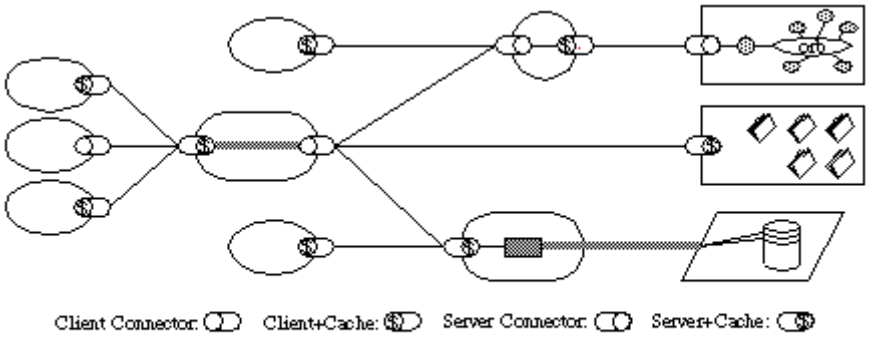

Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

A principal desvantagem dos sistemas em camadas é que eles adicionam sobrecarga e latência ao processamento de dados, reduzindo o desempenho percebido pelo usuário [32]. Para um sistema baseado em rede que suporta restrições de cache, isso pode ser compensado pelos benefícios do cache compartilhado em intermediários. Colocar caches compartilhados nos limites de um domínio organizacional pode resultar em benefícios significativos de desempenho [136]. Essas camadas também permitem que as políticas de segurança sejam aplicadas aos dados que cruzam a fronteira organizacional, conforme exigido pelos firewalls [79].

A combinação de sistema em camadas e restrições de interface uniforme induz propriedades arquitetônicas semelhantes às do estilo tubo e filtro uniforme (Seção 3.2.2). Embora a interação REST seja bidirecional, os fluxos de dados de grande granularidade da interação hipermídia podem ser processados como uma rede de fluxo de dados, com componentes de filtro aplicados seletivamente ao fluxo de dados para transformar o conteúdo à medida que ele passa [26]. Dentro do REST, os componentes intermediários podem transformar

ativamente o conteúdo das mensagens porque as mensagens são autodescritivas e sua semântica é visível para os intermediários.

5.1.7 Código sob demanda

A adição final ao nosso conjunto de restrições para REST vem do estilo de código sob demanda da Seção 3.5.3 (Figura 5-8). REST permite que a funcionalidade do cliente seja estendida baixando e executando código na forma de applets ou scripts. Isso simplifica os clientes, reduzindo o número de recursos necessários para serem pré-implementados. Permitir que os recursos sejam baixados após a implantação melhora a extensibilidade do sistema. No entanto, também reduz a visibilidade e, portanto, é apenas uma restrição opcional no REST.

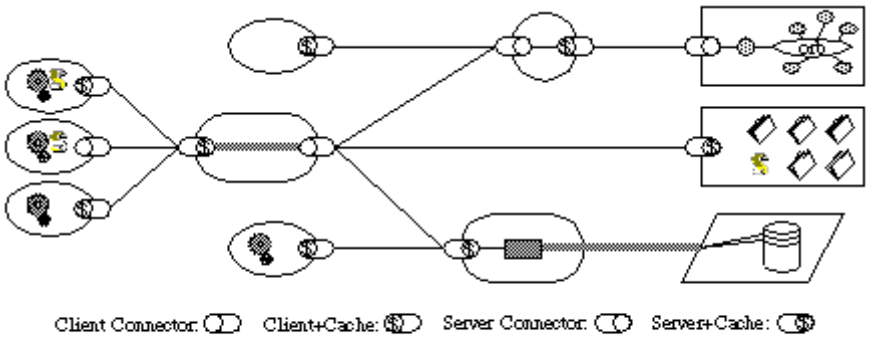

Figure 5-8. REST

A noção de uma restrição opcional pode parecer um oxímoro. No entanto, ele tem um propósito no projeto de arquitetura de um sistema que abrange vários limites organizacionais. Isso significa que a arquitetura só ganha o benefício (e sofre as desvantagens) das restrições opcionais quando elas são conhecidas por estarem em vigor para algum domínio do sistema geral. Por exemplo, se todo o software cliente dentro de uma organização é conhecido por suportar applets Java [45], então os serviços dentro dessa organização podem ser construídos de forma que obtenham o benefício de funcionalidade aprimorada por meio de classes Java para download. Ao mesmo tempo, no entanto, o firewall da organização pode impedir a transferência de applets Java de fontes externas e, assim, para o resto da Web parecerá que esses clientes não suportam código sob demanda. Uma restrição opcional nos permite projetar uma arquitetura que suporte o comportamento desejado no caso geral, mas com o entendimento de que pode ser desabilitada em alguns contextos.

5.1.8 Resumo de Derivação de Estilo

REST consiste em um conjunto de restrições arquitetônicas escolhidas para as propriedades que elas induzem nas arquiteturas candidatas. Embora cada uma dessas restrições possa ser considerada isoladamente, descrevê-las em termos de sua derivação de estilos arquitetônicos comuns facilita a compreensão do raciocínio por trás de sua seleção. A Figura 5-9 mostra a derivação das restrições do REST graficamente em termos dos estilos de arquitetura baseados em rede examinados no Capítulo 3.

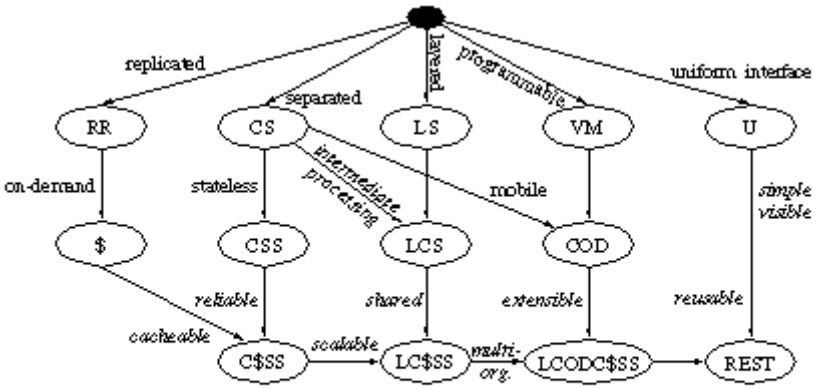

Figure 5-9. REST Derivation by Style Constraints

5.2 Elementos Arquitetônicos REST

O estilo Representational State Transfer (REST) é uma abstração dos elementos arquitetônicos dentro de um sistema de hipermídia distribuído. O REST ignora os detalhes da implementação do componente e da sintaxe do protocolo para se concentrar nas funções dos componentes, nas restrições sobre sua interação com outros componentes e na interpretação de elementos de dados significativos. Ele abrange as restrições fundamentais sobre componentes, conectores e dados que definem a base da arquitetura da Web e, portanto, a essência de seu comportamento como um aplicativo baseado em rede.

5.2.1 Elementos de Dados

Ao contrário do estilo de objeto distribuído [31], onde todos os dados são encapsulados e ocultos pelos componentes de processamento, a natureza e o estado dos elementos de dados de uma arquitetura é um aspecto chave do REST. A razão para este projeto pode ser vista na natureza da hipermídia distribuída. Quando um link é selecionado, a informação precisa ser movida do local onde está armazenada para o local onde será usada, na maioria dos casos, por um leitor humano. Isso é diferente de muitos outros paradigmas de processamento distribuído [6 , 50], onde é possível, e geralmente mais eficiente, mover o "agente de processamento" (por exemplo, código móvel, procedimento armazenado, expressão de pesquisa, etc.) do que mover os dados para o processador.

Um arquiteto de hipermídia distribuído tem apenas três opções fundamentais: 1) renderizar os dados onde estão localizados e enviar uma imagem de formato fixo ao destinatário; 2) encapsular os dados com um mecanismo de renderização e enviar ambos ao destinatário; ou, 3) enviar os dados brutos para o destinatário junto com os metadados que descrevem o tipo de dados, para que o destinatário possa escolher seu próprio mecanismo de renderização.

Cada opção tem suas vantagens e desvantagens. A opção 1, o estilo tradicional cliente-servidor [31], permite que todas as informações sobre a verdadeira natureza dos dados permaneçam ocultas no remetente, evitando que sejam feitas suposições sobre a estrutura dos dados e facilitando a implementação do cliente. No entanto, também restringe severamente a funcionalidade do destinatário e coloca a maior parte da carga de processamento no remetente, levando a problemas de escalabilidade. Opção 2, o estilo de objeto móvel [50], fornece ocultação de informações enquanto permite o processamento especializado dos dados por meio de seu mecanismo de renderização exclusivo, mas limita a funcionalidade do destinatário ao que é previsto nesse mecanismo e pode aumentar muito a quantidade de dados transferidos. A opção 3 permite que o remetente permaneça simples e escalável, minimizando os bytes transferidos, mas perde as vantagens da ocultação de informações e exige que o remetente e o destinatário compreendam os mesmos tipos de dados.

O REST fornece um híbrido de todas as três opções, concentrando-se em um entendimento compartilhado de tipos de dados com metadados, mas limitando o escopo do que é revelado a uma interface padronizada. Os componentes REST se comunicam transferindo uma representação de um recurso em um formato correspondente a um conjunto de tipos de dados padrão em evolução, selecionados dinamicamente com base nas capacidades ou desejos do destinatário e na natureza do recurso. Se a representação está no mesmo

formato que a fonte bruta ou é derivada da fonte, permanece oculto atrás da interface. Os benefícios do estilo de objeto móvel são aproximados enviando uma representação que consiste em instruções no formato de dados padrão de um mecanismo de renderização encapsulado (por exemplo, Java [45]). O REST, portanto, ganha a separação de preocupações do estilo cliente-servidor sem o problema de escalabilidade do servidor, permite ocultar informações por meio de uma interface genérica para permitir o encapsulamento e a evolução de serviços e fornece um conjunto diversificado de funcionalidades por meio de mecanismos de recursos para download.

Os elementos de dados do REST estão resumidos na Tabela 5-1 .

Tabela 5-1: Elementos de dados REST

Elemento de dados	Exemplos da Web Moderna
recurso	o alvo conceitual pretendido de uma referência de hipertexto
identificador de recurso	URL, URL
representação	Documento HTML, imagem JPEG
metadados de representação	tipo de mídia, hora da última modificação
metadados de recursos	link de origem, alternativas, variar
dados de controle	if-modified-desde, controle de cache

5.2.1.1 Recursos e Identificadores de Recursos

A abstração chave de informações em REST é um *recurso* . Qualquer informação que possa ser nomeada pode ser um recurso: um documento ou imagem, um serviço temporal (por exemplo, "tempo de hoje em Los Angeles"), uma coleção de outros recursos, um objeto não virtual (por exemplo, uma pessoa) e assim por diante . Em outras palavras, qualquer conceito que possa ser alvo de referência de hipertexto de um autor deve se encaixar na definição de um recurso. Um recurso é um mapeamento conceitual para um conjunto de entidades, não a entidade que corresponde ao mapeamento em um determinado momento.

Mais precisamente, um recurso *R é uma função de pertinência* $M_R(t)$ que varia temporalmente , que para o tempo *t* mapeia para um conjunto de entidades, ou valores, que são equivalentes. Os valores no conjunto podem ser *representações de recursos* e/ou *identificadores de recursos* . Um recurso pode mapear para o conjunto vazio, o que permite que sejam feitas referências a um conceito antes que qualquer realização desse conceito exista - uma noção que era estranha à maioria dos sistemas de hipertexto anteriores à Web [61]. Alguns recursos são estáticos no sentido de que, quando examinados a qualquer momento após a sua criação, correspondem sempre ao mesmo conjunto de valores. Outros têm um alto grau de variação em seu valor ao longo do tempo. A única coisa que precisa ser estática para um recurso é a semântica do mapeamento, já que a semântica é o que diferencia um recurso do outro.

Por exemplo, a "versão preferida dos autores" de um artigo acadêmico é um mapeamento cujo valor muda ao longo do tempo, enquanto um mapeamento para "o artigo publicado nos anais da conferência X" é estático. Esses são dois recursos distintos, mesmo que ambos sejam mapeados para o mesmo valor em algum momento. A distinção é necessária para que ambos os recursos possam ser identificados e referenciados de forma independente. Um exemplo semelhante da engenharia de software é a identificação separada de um arquivo de código-fonte controlado por versão ao se referir à "revisão mais recente", "número de revisão 1.2.7" ou "revisão incluída na versão Orange".

Essa definição abstrata de um recurso habilita os principais recursos da arquitetura da Web. Primeiro, ele fornece generalidade ao abranger muitas fontes de informação sem distingui-las artificialmente por tipo ou implementação. Em segundo lugar, permite a vinculação tardia da referência a uma representação, permitindo que a negociação de conteúdo ocorra com base nas características da solicitação. Finalmente, permite que um autor faça referência ao conceito em vez de uma representação singular desse conceito, eliminando assim a necessidade de alterar todos os links existentes sempre que a representação mudar (assumindo que o autor usou o identificador correto).

REST usa um identificador de recurso para identificar o recurso específico envolvido em uma interação entre componentes. Os conectores REST fornecem uma interface genérica para acessar e manipular o conjunto de

valores de um recurso, independentemente de como a função de associação é definida ou do tipo de software que está processando a solicitação. A autoridade de nomenclatura que atribuiu o identificador de recurso, tornando possível fazer referência ao recurso, é responsável por manter a validade semântica do mapeamento ao longo do tempo (ou seja, garantir que a função de associação não seja alterada).

Sistemas de hipertexto tradicionais [61], que normalmente operam em um ambiente fechado ou local, usam identificadores únicos de nó ou documento que mudam toda vez que a informação muda, contando com servidores de link para manter referências separadas do conteúdo [135]. Como os servidores de link centralizados são um anátema para a imensa escala e os requisitos de domínio multiorganizacional da Web, o REST depende da escolha do autor de um identificador de recurso que melhor se ajuste à natureza do conceito que está sendo identificado. Naturalmente, a qualidade de um identificador é muitas vezes proporcional à quantidade de dinheiro gasto para manter sua validade, o que leva a links quebrados à medida que informações efêmeras (ou mal suportadas) se movem ou desaparecem com o tempo.

5.2.1.2 Representações

Os componentes REST executam ações em um recurso usando uma representação para capturar o estado atual ou pretendido desse recurso e transferindo essa representação entre os componentes. Uma representação é uma sequência de bytes, mais metadados de representação para descrever esses bytes. Outros nomes comumente usados, mas menos precisos, para uma representação incluem: documento, arquivo e entidade, instância ou variante de mensagem HTTP.

Uma representação consiste em dados, metadados que descrevem os dados e, ocasionalmente, metadados para descrever os metadados (geralmente com a finalidade de verificar a integridade da mensagem). Os metadados estão na forma de pares nome-valor, onde o nome corresponde a um padrão que define a estrutura e a semântica do valor. As mensagens de resposta podem incluir metadados de representação e metadados de recurso: informações sobre o recurso que não são específicas da representação fornecida.

Os dados de controle definem a finalidade de uma mensagem entre os componentes, como a ação solicitada ou o significado de uma resposta. Também é usado para parametrizar solicitações e substituir o comportamento padrão de alguns elementos de conexão. Por exemplo, o comportamento do cache pode ser modificado pelos dados de controle incluídos na mensagem de solicitação ou resposta.

Dependendo dos dados de controle da mensagem, uma determinada representação pode indicar o estado atual do recurso solicitado, o estado desejado para o recurso solicitado ou o valor de algum outro recurso, como uma representação dos dados de entrada no formulário de consulta de um cliente, ou uma representação de alguma condição de erro para uma resposta. Por exemplo, a autoria remota de um recurso requer que o autor envie uma representação ao servidor, estabelecendo assim um valor para aquele recurso que pode ser recuperado por solicitações posteriores. Se o conjunto de valores de um recurso em um determinado momento consiste em várias representações, a negociação de conteúdo pode ser usada para selecionar a melhor representação para inclusão em uma determinada mensagem.

O formato de dados de uma representação é conhecido como tipo de mídia [48]. Uma representação pode ser incluída em uma mensagem e processada pelo destinatário de acordo com os dados de controle da mensagem e a natureza do tipo de mídia. Alguns tipos de mídia destinam-se ao processamento automatizado, alguns devem ser renderizados para visualização por um usuário e alguns são capazes de ambos. Os tipos de mídia composta podem ser usados para incluir várias representações em uma única mensagem.

O design de um tipo de mídia pode afetar diretamente o desempenho percebido pelo usuário de um sistema de hipermídia distribuído. Quaisquer dados que devam ser recebidos antes que o destinatário possa começar a renderizar a representação aumentam a latência de uma interação. Um formato de dados que coloca as informações de renderização mais importantes na frente, de modo que as informações iniciais possam ser renderizadas de forma incremental enquanto o restante das informações está sendo recebido, resulta em um desempenho percebido pelo usuário muito melhor do que um formato de dados que deve ser recebido inteiramente antes renderização pode começar.

Por exemplo, um navegador da Web que pode renderizar de forma incremental um documento HTML grande enquanto ele está sendo recebido oferece desempenho percebido pelo usuário significativamente melhor do

que um que espera até que todo o documento seja recebido completamente antes da renderização, mesmo que o desempenho da rede seja o mesmo. Observe que a capacidade de renderização de uma representação também pode ser afetada pela escolha do conteúdo. Se as dimensões de tabelas de tamanho dinâmico e objetos incorporados precisarem ser determinadas antes de serem renderizadas, sua ocorrência na área de visualização de uma página hipermídia aumentará sua latência.

5.2.2 Conectores

REST usa vários tipos de conectores, resumidos na Tabela 5-2 , para encapsular as atividades de acesso a recursos e transferência de representações de recursos. Os conectores apresentam uma interface abstrata para comunicação de componentes, aumentando a simplicidade ao fornecer uma separação limpa de interesses e ocultando a implementação subjacente de recursos e mecanismos de comunicação. A generalidade da interface também permite a substituibilidade: se o único acesso dos usuários ao sistema for por meio de uma interface abstrata, a implementação pode ser substituída sem impactar os usuários. Como um conector gerencia a comunicação de rede para um componente, as informações podem ser compartilhadas em várias interações para melhorar a eficiência e a capacidade de resposta.

Tabela 5-2: Conectores REST

Conector	Exemplos da Web Moderna
cliente	libwww, libwww-perl
servidor	libwww, Apache API, NSAPI
esconderijo	cache do navegador, rede de cache Akamai
resolver	bind (biblioteca de pesquisa DNS)
túnel	SOCKS, SSL após HTTP CONNECT

Todas as interações REST são sem estado. Ou seja, cada solicitação contém todas as informações necessárias para que um conector entenda a solicitação, independentemente de quaisquer solicitações que possam tê-la precedido. Essa restrição cumpre quatro funções: 1) elimina a necessidade de os conectores manterem o estado da aplicação entre as requisições, reduzindo assim o consumo de recursos físicos e melhorando a escalabilidade; 2) permite que as interações sejam processadas em paralelo sem exigir que o mecanismo de processamento entenda a semântica da interação; 3) permite que um intermediário visualize e entenda uma solicitação isoladamente, o que pode ser necessário quando os serviços são reorganizados dinamicamente; e, 4) força todas as informações que podem influenciar na reutilização de uma resposta em cache a estarem presentes em cada solicitação.

A interface do conector é semelhante à chamada procedural, mas com diferenças importantes na passagem de parâmetros e resultados. Os in-parameters consistem em dados de controle de solicitação, um identificador de recurso que indica o destino da solicitação e uma representação opcional. Os parâmetros out consistem em dados de controle de resposta, metadados de recursos opcionais e uma representação opcional. De um ponto de vista abstrato, a invocação é síncrona, mas os parâmetros de entrada e saída podem ser passados como fluxos de dados. Em outras palavras, o processamento pode ser invocado antes que o valor dos parâmetros seja completamente conhecido, evitando assim a latência do processamento em lote de grandes transferências de dados.

Os tipos de conector primário são cliente e servidor. A diferença essencial entre os dois é que um cliente inicia a comunicação fazendo uma solicitação, enquanto um servidor escuta as conexões e responde às solicitações para fornecer acesso aos seus serviços. Um componente pode incluir conectores de cliente e servidor.

Um terceiro tipo de conector, o conector de cache, pode ser localizado na interface para um conector de cliente ou servidor para salvar as respostas que podem ser armazenadas em cache às interações atuais para que possam ser reutilizadas para interações solicitadas posteriormente. Um cache pode ser usado por um cliente para evitar a repetição da comunicação da rede, ou por um servidor para evitar a repetição do processo de geração de uma resposta, com ambos os casos servindo para reduzir a latência da interação. Normalmente, um cache é implementado no espaço de endereço do conector que o utiliza.

Alguns conectores de cache são compartilhados, o que significa que suas respostas em cache podem ser usadas em resposta a um cliente diferente daquele para o qual a resposta foi originalmente obtida. O cache compartilhado pode ser eficaz na redução do impacto de "flash crowds" na carga de um servidor popular, principalmente quando o cache é organizado hierarquicamente para cobrir grandes grupos de usuários, como aqueles dentro da intranet de uma empresa, os clientes de um serviço de Internet provedor, ou Universidades que compartilham um backbone de rede nacional. No entanto, o cache compartilhado também pode levar a erros se a resposta em cache não corresponder ao que teria sido obtido por uma nova solicitação. O REST tenta equilibrar o desejo de transparência no comportamento do cache com o desejo de uso eficiente da rede, em vez de assumir que a transparência absoluta é sempre necessária.

Um cache é capaz de determinar a capacidade de cache de uma resposta porque a interface é genérica e não específica para cada recurso. Por padrão, a resposta a uma solicitação de recuperação pode ser armazenada em cache e as respostas a outras solicitações não podem ser armazenadas em cache. Se alguma forma de autenticação do usuário fizer parte da solicitação ou se a resposta indicar que ela não deve ser compartilhada, a resposta só poderá ser armazenada em cache por um cache não compartilhado. Um componente pode substituir esses padrões incluindo dados de controle que marcam a interação como armazenável em cache, não armazenável em cache ou armazenável em cache apenas por um tempo limitado.

Um resolvedor converte identificadores de recursos parciais ou completos nas informações de endereço de rede necessárias para estabelecer uma conexão entre componentes. Por exemplo, a maioria dos URIs inclui um nome de host DNS como mecanismo para identificar a autoridade de nomenclatura do recurso. Para iniciar uma solicitação, um navegador da Web extrairá o nome do host do URI e usará um resolvedor de DNS para obter o endereço de protocolo da Internet para essa autoridade. Outro exemplo é que alguns esquemas de identificação (por exemplo, URN [124]) exigem que um intermediário traduza um identificador permanente para um endereço mais transitório para acessar o recurso identificado. O uso de um ou mais resolvedores intermediários pode melhorar a longevidade das referências de recursos por meio de indireção, embora isso aumente a latência da solicitação.

A forma final do tipo de conector é um túnel, que simplesmente retransmite a comunicação através de um limite de conexão, como um firewall ou gateway de rede de nível inferior. A única razão pela qual ele é modelado como parte do REST e não abstraído como parte da infraestrutura de rede é que alguns componentes REST podem alternar dinamicamente do comportamento do componente ativo para o de um túnel. O exemplo principal é um proxy HTTP que muda para um túnel em resposta a uma solicitação do método CONNECT [71], permitindo assim que seu cliente se comunique diretamente com um servidor remoto usando um protocolo diferente, como TLS, que não permite proxies. O túnel desaparece quando ambas as extremidades terminam sua comunicação.

5.2.3 Componentes

Os componentes REST, resumidos na Tabela 5-3 , são tipados por suas funções em uma ação geral do aplicativo.

Tabela 5-3: Componentes REST

Componente	Exemplos da Web Moderna
servidor de origem	Apache httpd, Microsoft IIS
Porta de entrada	Lula, CGI, Proxy Reverso
procurador	Proxy CERN, Proxy Netscape, Gauntlet
agente de usuário	Netscape Navigator, Lynx, MOMspider

Um agente de usuário usa um conector de cliente para iniciar uma solicitação e se torna o destinatário final da resposta. O exemplo mais comum é um navegador da Web, que fornece acesso a serviços de informação e fornece respostas de serviço de acordo com as necessidades da aplicação.

Um servidor de origem usa um conector de servidor para controlar o namespace de um recurso solicitado. É a fonte definitiva de representação de seus recursos e deve ser o destinatário final de qualquer solicitação que pretenda modificar o valor de seus recursos. Cada servidor de origem fornece uma interface genérica para

seus serviços como uma hierarquia de recursos. Os detalhes da implementação do recurso estão ocultos atrás da interface.

Os componentes intermediários atuam como cliente e servidor para encaminhar, com possível tradução, solicitações e respostas. Um componente proxy é um intermediário selecionado por um cliente para fornecer encapsulamento de interface de outros serviços, tradução de dados, aprimoramento de desempenho ou proteção de segurança. Um componente de gateway (também conhecido como proxy reverso) é um intermediário imposto pela rede ou servidor de origem para fornecer um encapsulamento de interface de outros serviços, para tradução de dados, aprimoramento de desempenho ou aplicação de segurança. Observe que a diferença entre um proxy e um gateway é que um cliente determina quando usará um proxy.

5.3 Visões arquitetônicas REST

Agora que temos uma compreensão dos elementos arquiteturais REST isoladamente, podemos usar visualizações arquiteturais [105] para descrever como os elementos trabalham juntos para formar uma arquitetura. Três tipos de visualização – processo, conector e dados – são úteis para iluminar os princípios de design do REST.

5.3.1 Visualização do Processo

Uma visão de processo de uma arquitetura é principalmente eficaz para obter os relacionamentos de interação entre os componentes, revelando o caminho dos dados à medida que eles fluem pelo sistema. Infelizmente, a interação de um sistema real geralmente envolve um grande número de componentes, resultando em uma visão geral obscurecida pelos detalhes. A Figura 5-10 fornece uma amostra da visualização do processo de uma arquitetura baseada em REST em uma instância específica durante o processamento de três solicitações paralelas.

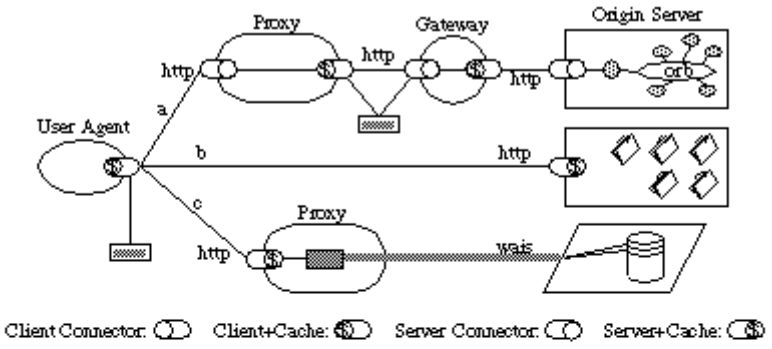

Figure 5-10. Process View of a REST-based Architecture

A user agent is portrayed in the midst of three parallel interactions: a, b, and c. The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector. Request (a) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture. Request (b) is sent directly to an origin server, which is able to satisfy the request from its own cache. Request (c) is sent to a proxy that is capable of directly accessing WAIS, an information service that is separate from the Web architecture, and translating the WAIS response into a format recognized by the generic connector interface. Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view.

A separação de interesses cliente-servidor do REST simplifica a implementação de componentes, reduz a complexidade da semântica do conector, melhora a eficácia do ajuste de desempenho e aumenta a escalabilidade de componentes de servidor puros. As restrições do sistema em camadas permitem que intermediários - proxies, gateways e firewalls - sejam introduzidos em vários pontos da comunicação sem alterar as interfaces entre os componentes, permitindo que eles ajudem na tradução da comunicação ou melhorem o desempenho por meio de cache compartilhado em grande escala. O REST permite o processamento intermediário restringindo as mensagens a serem autodescritivas: a interação não tem estado entre as solicitações, os métodos padrão e os tipos de mídia são usados para indicar a semântica e trocar informações, e as respostas indicam explicitamente a capacidade de cache.

Como os componentes são conectados dinamicamente, sua disposição e função para uma ação de aplicativo específica tem características semelhantes a um estilo de tubulação e filtro. Embora os componentes REST se comuniquem por meio de fluxos bidirecionais, o processamento de cada direção é independente e, portanto, suscetível a transdutores de fluxo (filtros). A interface do conector genérico permite que os componentes sejam colocados no fluxo com base nas propriedades de cada solicitação ou resposta.

Os serviços podem ser implementados usando uma hierarquia complexa de intermediários e vários servidores de origem distribuídos. A natureza stateless do REST permite que cada interação seja independente das outras, eliminando a necessidade de uma conscientização da topologia geral dos componentes, uma tarefa impossível para uma arquitetura em escala de Internet, e permitindo que os componentes atuem como destinos ou intermediários, determinados dinamicamente. pelo destino de cada solicitação. Os conectores só precisam estar cientes da existência um do outro durante o escopo de sua comunicação, embora possam armazenar em cache a existência e os recursos de outros componentes por motivos de desempenho.

5.3.2 Visualização do Conector

Uma visão de conector de uma arquitetura concentra-se na mecânica da comunicação entre os componentes. Para uma arquitetura baseada em REST, estamos particularmente interessados nas restrições que definem a interface de recursos genéricos.

Os conectores de cliente examinam o identificador de recurso para selecionar um mecanismo de comunicação apropriado para cada solicitação. Por exemplo, um cliente pode ser configurado para se conectar a um componente proxy específico, talvez um agindo como um filtro de anotação, quando o identificador indica que é um recurso local. Da mesma forma, um cliente pode ser configurado para rejeitar solicitações para algum subconjunto de identificadores.

O REST não restringe a comunicação a um protocolo específico, mas restringe a interface entre os componentes e, portanto, o escopo das suposições de interação e implementação que poderiam ser feitas entre os componentes. Por exemplo, o protocolo de transferência primário da Web é HTTP, mas a arquitetura também inclui acesso contínuo a recursos que se originam em servidores de rede pré-existentes, incluindo FTP [107], Gopher [7] e WAIS [36]]. A interação com esses serviços é restrita à semântica de um conector REST. Essa restrição sacrifica algumas das vantagens de outras arquiteturas, como a interação com estado de um protocolo de realimentação de relevância como WAIS, a fim de manter as vantagens de uma única interface genérica para semântica de conectores. Em contrapartida, a interface genérica possibilita o acesso a uma infinidade de serviços por meio de um único proxy. Se um aplicativo precisar de recursos adicionais de outra arquitetura, ele poderá implementar e invocar esses recursos como um sistema separado executado em paralelo, semelhante à forma como a arquitetura da Web interage com os recursos "telnet" e "mailto".

5.3.3 Visualização de Dados

Uma visualização de dados de uma arquitetura revela o estado do aplicativo à medida que as informações fluem pelos componentes. Como o REST é especificamente direcionado a sistemas de informação distribuídos, ele vê um aplicativo como uma estrutura coesa de informações e alternativas de controle por meio do qual um usuário pode executar uma tarefa desejada. Por exemplo, pesquisar uma palavra em um dicionário on-line é uma aplicação, assim como passear por um museu virtual ou revisar um conjunto de anotações de aula para estudar para um exame. Cada aplicativo define metas para o sistema subjacente, em relação às quais o desempenho do sistema pode ser medido.

As interações de componentes ocorrem na forma de mensagens de tamanho dinâmico. Mensagens de granulação pequena ou média são usadas para semântica de controle, mas a maior parte do trabalho do aplicativo é realizada por meio de mensagens de granulação grande contendo uma representação completa de recursos. A forma mais frequente de semântica de solicitação é a de recuperar uma representação de um recurso (por exemplo, o método "GET" em HTTP), que geralmente pode ser armazenado em cache para reutilização posterior.

REST concentra todo o estado de controle nas representações recebidas em resposta às interações. O objetivo é melhorar a escalabilidade do servidor eliminando qualquer necessidade de o servidor manter um conhecimento do estado do cliente além da solicitação atual. O estado de um aplicativo é, portanto, definido por suas solicitações pendentes, a topologia dos componentes conectados (alguns dos quais podem estar filtrando dados em buffer), as solicitações ativas nesses conectores, o fluxo de dados das representações em resposta a essas solicitações e o processamento dessas representações à medida que são recebidas pelo agente do usuário.

Um aplicativo atinge um estado estável sempre que não tiver solicitações pendentes; ou seja, ele não possui solicitações pendentes e todas as respostas ao seu conjunto atual de solicitações foram completamente recebidas ou recebidas até o ponto em que podem ser tratadas como um fluxo de dados de representação. Para um aplicativo de navegador, esse estado corresponde a uma "página da Web", incluindo a representação primária e as representações auxiliares, como imagens em linha, applets incorporados e folhas de estilo. A importância dos estados estáveis do aplicativo é vista em seu impacto no desempenho percebido pelo usuário e na intermitência do tráfego de solicitação de rede.

O desempenho percebido pelo usuário de um aplicativo de navegador é determinado pela latência entre os estados estáveis: o período de tempo entre a seleção de um link de hipermídia em uma página da Web e o ponto em que as informações úteis foram renderizadas para a próxima página da Web. A otimização do desempenho do navegador é, portanto, centrada na redução dessa latência de comunicação.

Como as arquiteturas baseadas em REST se comunicam principalmente por meio da transferência de representações de recursos, a latência pode ser afetada tanto pelo design dos protocolos de comunicação quanto pelo design dos formatos de dados de representação. A capacidade de renderizar incrementalmente os dados de resposta à medida que são recebidos é determinada pelo design do tipo de mídia e pela disponibilidade de informações de layout (dimensões visuais de objetos em linha) em cada representação.

Uma observação interessante é que a solicitação de rede mais eficiente é aquela que não usa a rede. Em outras palavras, a capacidade de reutilizar uma resposta em cache resulta em uma melhoria considerável no desempenho do aplicativo. Embora o uso de um cache adicione alguma latência a cada solicitação individual devido à sobrecarga de pesquisa, a latência média de solicitação é significativamente reduzida quando mesmo uma pequena porcentagem de solicitações resulta em acertos de cache utilizáveis.

O próximo estado de controle de um aplicativo reside na representação do primeiro recurso solicitado, portanto, obter essa primeira representação é uma prioridade. A interação REST é, portanto, aprimorada por protocolos que "respondem primeiro e pensam depois". Em outras palavras, um protocolo que requer múltiplas interações por ação do usuário, a fim de fazer coisas como negociar recursos de recursos antes de enviar uma resposta de conteúdo, será perceptivelmente mais lento do que um protocolo que envia o que for mais provável que seja o ideal primeiro e depois fornece uma lista de alternativas para o cliente recuperar se a primeira resposta for insatisfatória.

O estado do aplicativo é controlado e armazenado pelo agente do usuário e pode ser composto por representações de vários servidores. Além de liberar o servidor dos problemas de escalabilidade de armazenamento de estado, isso permite que o usuário manipule diretamente o estado (por exemplo, o histórico de um navegador da Web), antecipe alterações nesse estado (por exemplo, mapas de links e pré-busca de representações) e salte de um aplicativo para outro (por exemplo, marcadores e caixas de diálogo de entrada de URI).

A aplicação do modelo é, portanto, um mecanismo que se move de um estado para o próximo examinando e escolhendo entre as transições de estado alternativas no conjunto atual de representações. Não surpreendentemente, isso corresponde exatamente à interface do usuário de um navegador de hipermídia. No entanto, o estilo não pressupõe que todos os aplicativos sejam navegadores. Na verdade, os detalhes do aplicativo são ocultos do servidor pela interface do conector genérico e, portanto, um agente de usuário pode ser igualmente um robô automatizado executando a recuperação de informações para um serviço de indexação, um agente pessoal procurando dados que correspondam a determinados critérios ou uma manutenção spider ocupado patrulhando as informações para referências quebradas ou conteúdo modificado [39].

5.4 Trabalho Relacionado

Bass, et ai. [9] dedicam um capítulo sobre arquitetura para a World Wide Web, mas sua descrição abrange apenas a arquitetura de implementação dentro do software libwww (bibliotecas cliente e servidor) desenvolvido pelo CERN/W3C e Jigsaw. Embora essas implementações reflitam muitas das restrições de design do REST, tendo sido desenvolvidas por pessoas familiarizadas com o design e a lógica da arquitetura da Web, a arquitetura WWW real é independente de qualquer implementação única. A Web moderna é definida por suas interfaces e protocolos padrão, não como essas interfaces e protocolos são implementados em um determinado software.

O estilo REST se baseia em muitos paradigmas de processos distribuídos preexistentes [6 , 50], protocolos de comunicação e campos de software. As interações do componente REST são estruturadas em um estilo cliente-servidor em camadas, mas as restrições adicionais da interface de recursos genéricos criam a oportunidade de substituição e inspeção por intermediários. Solicitações e respostas têm a aparência de um estilo de chamada remota, mas as mensagens REST são direcionadas a um recurso conceitual em vez de um identificador de implementação.

Várias tentativas foram feitas para modelar a arquitetura da Web como uma forma de sistema de arquivos distribuído (por exemplo, WebNFS) ou como um sistema de objetos distribuídos [83]. No entanto, eles excluem vários tipos de recursos da Web ou estratégias de implementação como "não interessantes", quando na verdade sua presença invalida as suposições subjacentes a esses modelos. O REST funciona bem porque não limita a implementação de recursos a determinados modelos predefinidos, permitindo que cada aplicação escolha uma implementação que melhor atenda às suas próprias necessidades e possibilitando a substituição de implementações sem impactar o usuário.

O método de interação de enviar representações de recursos para componentes consumidores tem alguns paralelos com estilos de integração baseada em eventos (EBI). A principal diferença é que os estilos EBI são baseados em push. O componente que contém o estado (equivalente a um servidor de origem em REST) emite um evento sempre que o estado muda, independentemente de algum componente estar realmente interessado ou escutando tal evento. No estilo REST, os componentes de consumo geralmente puxam as representações. Embora isso seja menos eficiente quando visto como um único cliente que deseja monitorar um único recurso, a escala da Web torna inviável um modelo push não regulamentado.

O uso de princípios do estilo REST na Web, com sua noção clara de componentes, conectores e representações, está intimamente relacionado ao estilo arquitetural C2 [128]. O estilo C2 suporta o desenvolvimento de aplicativos distribuídos e dinâmicos, concentrando-se no uso estruturado de conectores para obter independência de substrato. Os aplicativos C2 contam com a notificação assíncrona de alterações de estado e mensagens de solicitação. Assim como outros esquemas baseados em eventos, o C2 é nominalmente baseado em push, embora uma arquitetura C2 possa operar no estilo pull do REST apenas emitindo uma notificação após o recebimento de uma solicitação. No entanto, o estilo C2 não possui as restrições amigáveis ao intermediário do REST, como a interface de recursos genéricos, interações sem estado garantidas e suporte intrínseco para armazenamento em cache.

5.5 Resumo

Este capítulo introduziu o estilo de arquitetura Representational State Transfer (REST) para sistemas hipermídia distribuídos. REST fornece um conjunto de restrições arquitetônicas que, quando aplicadas como um todo, enfatizam a escalabilidade das interações dos componentes, a generalidade das interfaces, a implantação independente dos componentes e os componentes intermediários para reduzir a latência da interação, reforçar a segurança e encapsular sistemas legados. Descrevi os princípios de engenharia de software que orientam o REST e as restrições de interação escolhidas para reter esses princípios, contrastando-os com as restrições de outros estilos de arquitetura.

O próximo capítulo apresenta uma avaliação da arquitetura REST por meio da experiência e das lições aprendidas com a aplicação do REST ao design, especificação e implantação da arquitetura Web moderna. Este trabalho incluiu a autoria das atuais especificações de rastreamento de padrões da Internet do

Hypertext Transfer Protocol (HTTP/1.1) e Uniform Resource Identifiers (URI), e a implementação da arquitetura através da biblioteca de protocolos do cliente libwww-perl e do servidor Apache HTTP.

[Início] [Anterior] [Próximo] *[Como referenciar este trabalho.]*