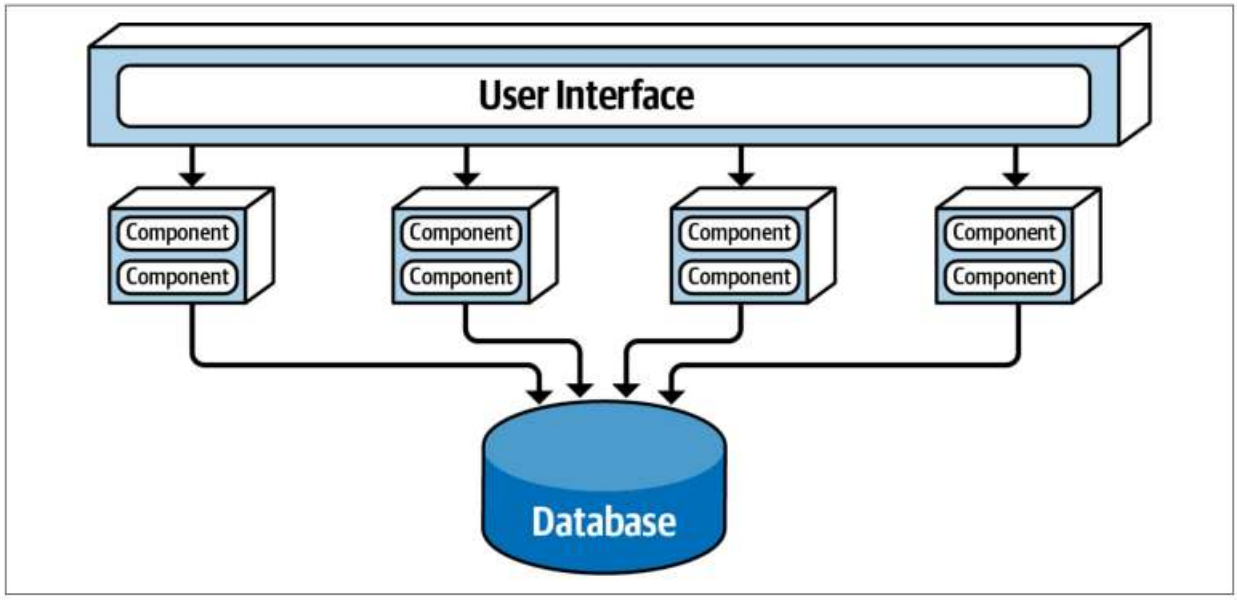
13 - Arquitetura Baseada em Serviços

2025-12-21 10:23 am

- Híbrido de arquitetura de microsserviços.
- Bem flexível.
- Distribuído mas sem muita complexidade de outros sistema distribuídos.

Topology



- *Figure 13-1. Basic topology of the service-based architecture style*
- Interface separada e serviços individuais não muito granulares, com database único.
    - Serviços de domínio.
- Cada serviço é deployado independentemente.
    - Costuma ter de 4 a 12 serviços.
- Tipicamente um serviço por domínio, mas requisitos podem mudar isso.
- Geralmente REST faz a comunicação entre interface e serviços
    - Mas pode ser RPC ou SOAP.
    - usa *Service Locator Pattern*
- Database único é importante.
    - Mudanças no database podem causar transtornos.

Variantes de Topologia

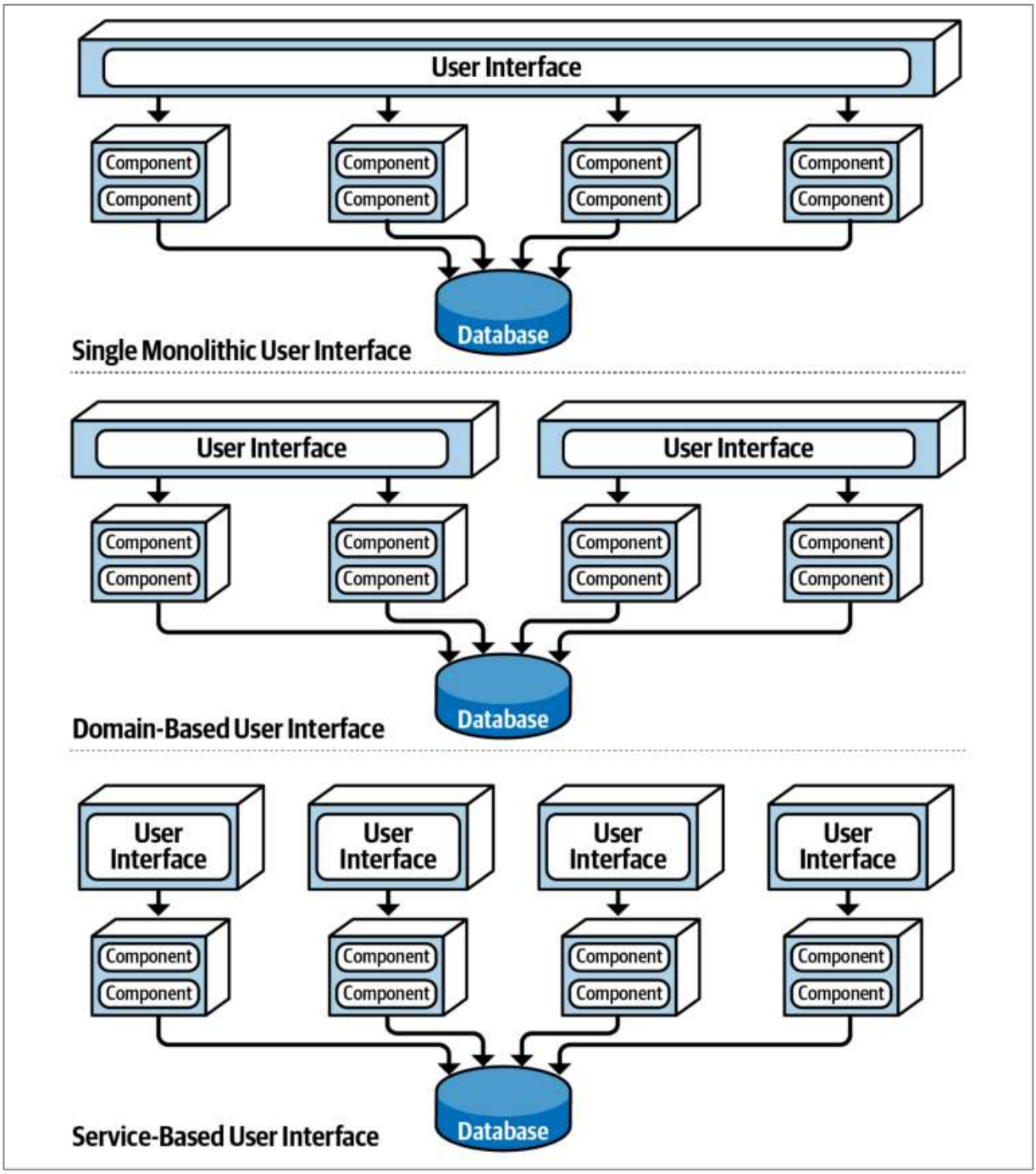


Figure 13-2. User interface variants

- Muitas variações, da forma de separar a interface do usuário.
- Também pode haver mais de um database, separados entre alguns serviços ou para cada serviço individual.
    - Requer garantir que um serviço não precisa dos dados do outro
- Também pode haver uma camada de API como um proxy reverso entre a interface e os serviços.

- **

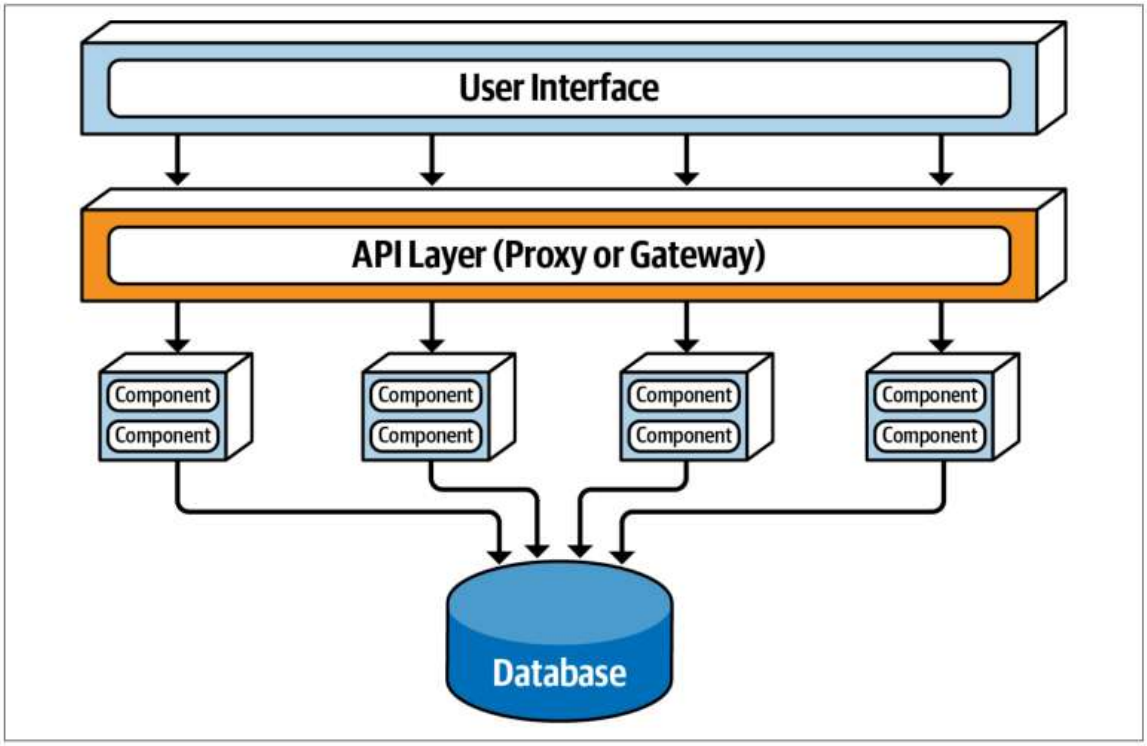


Figure 13-4. Adding an API layer between the user interface and domain services

- Boa prática para expor serviço externamente ou para separar responsabilidades (e.g. monitoramento)

Projeto do Serviço e Granularidade

- Serviços têm escopo mais ou menos largo.
- Cada serviço pode ter arquitetura em camadas.
- Também pode ser internamente separado por domínio.

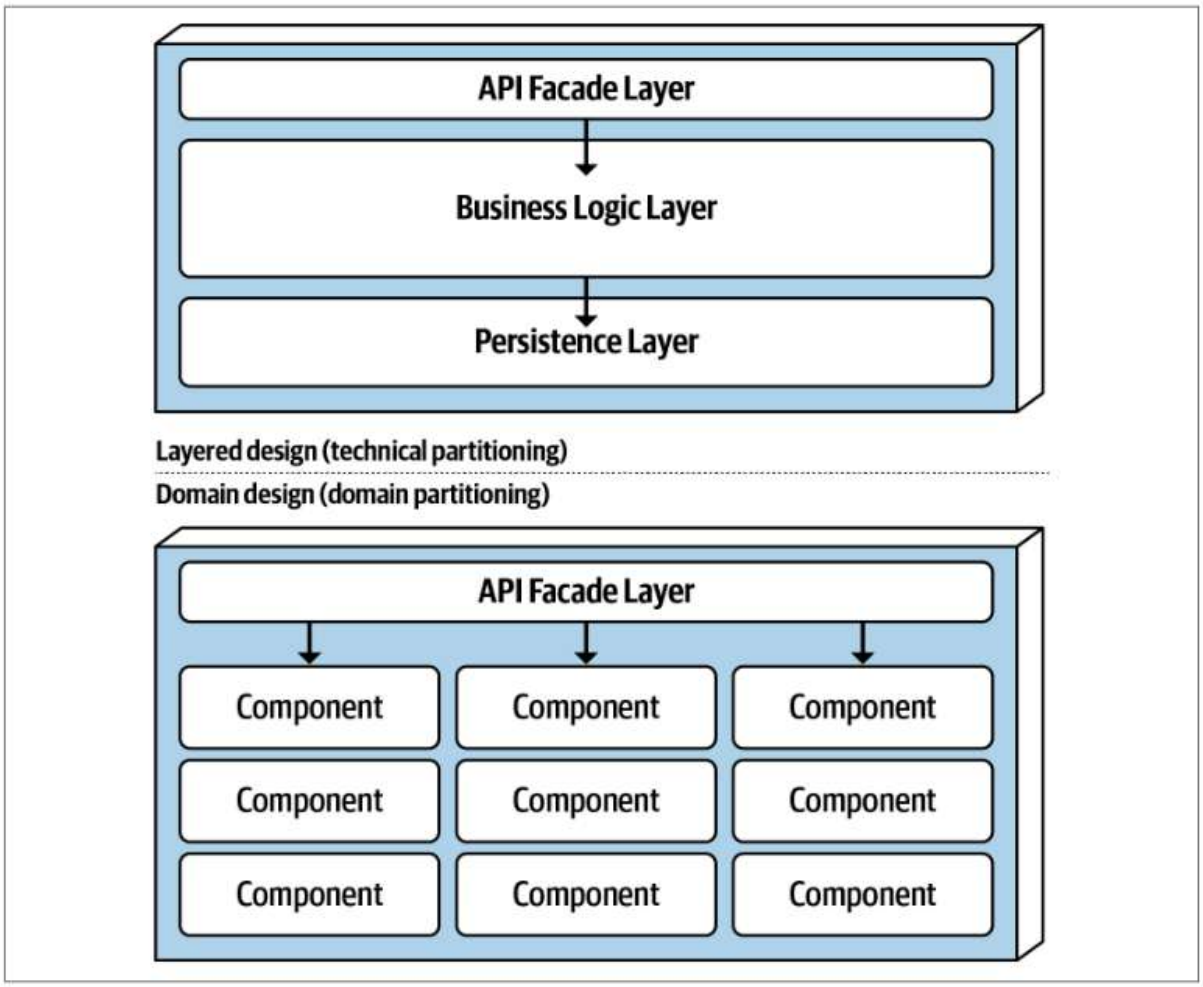
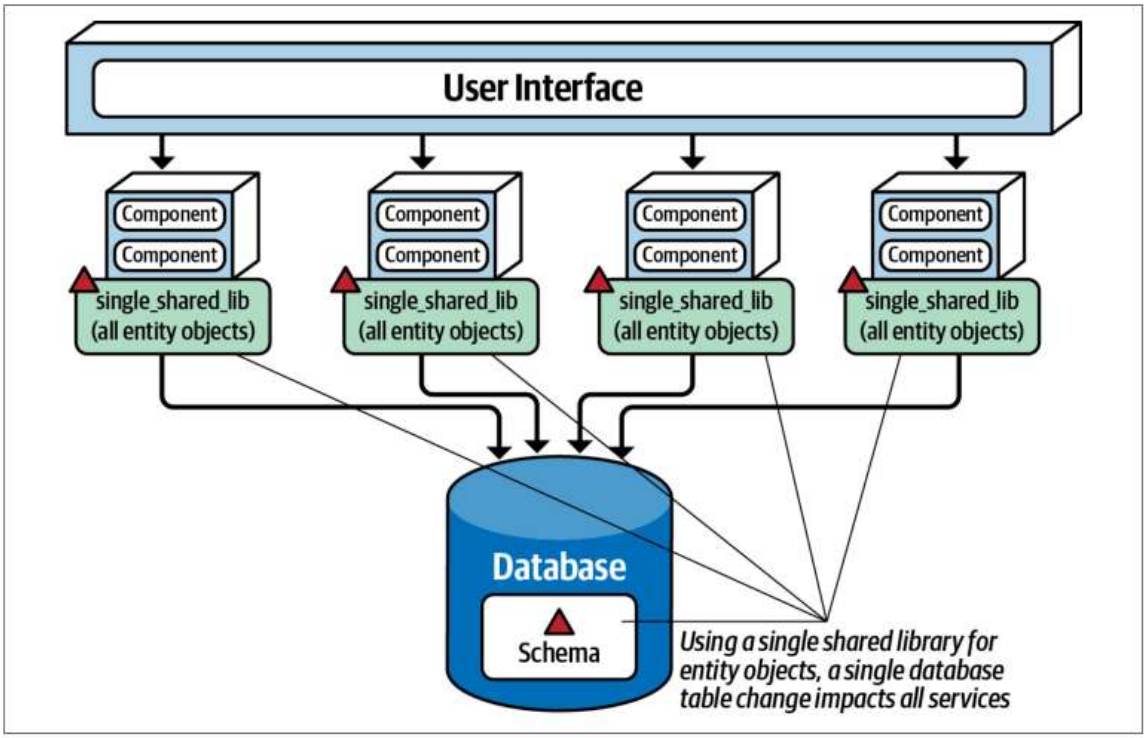


Figure 13-5. Domain service design variants

- Diferença com microsserviços é a granularidade dos serviços.
- Consegue garantir transações ACID.
    - Microsserviços costumam usar BASE (basic availability, soft state, consistência em algum momento)
- Mais fácil de dar rollback dentro de um mesmo serviço.
    - Serviços por domínio facilitam integridade dos dados.
    - Mas dificultam mudanças, já que uma mudança pequena dentro de um serviço requer o teste do serviço inteiro.

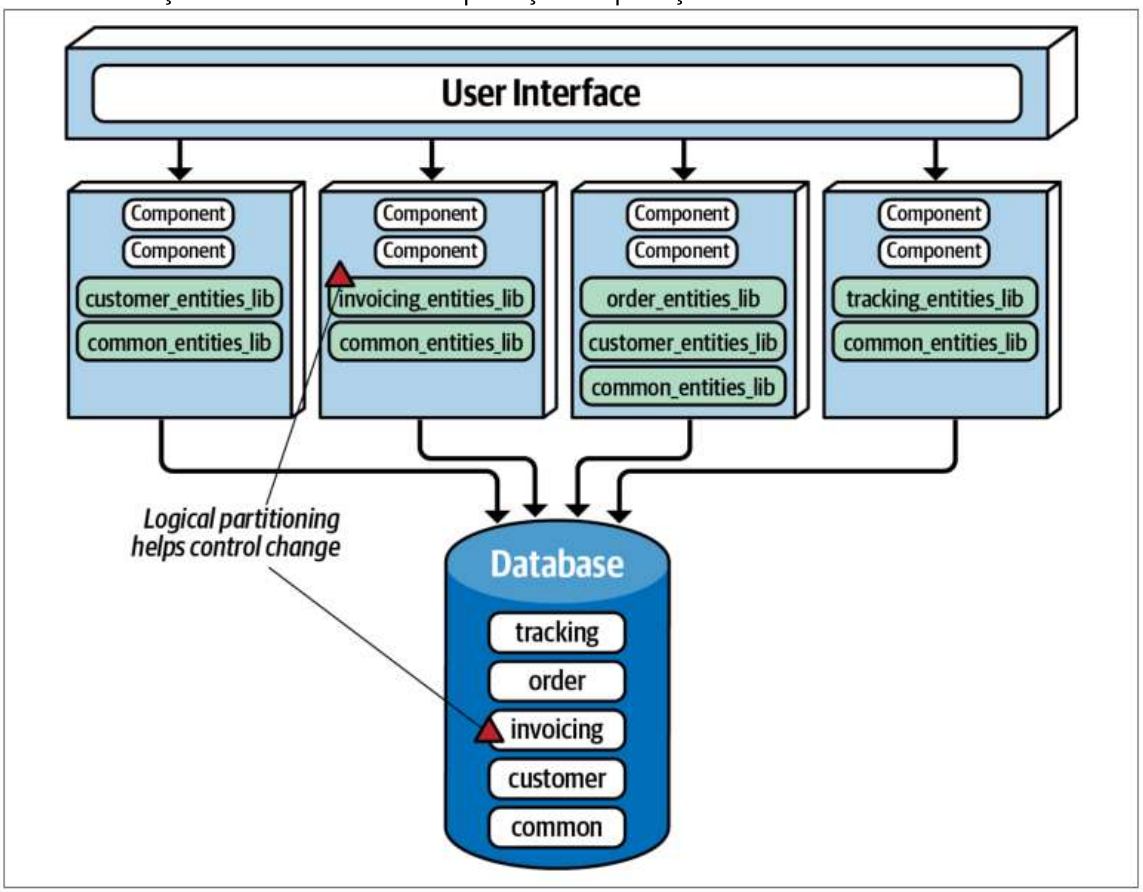
Partição de Banco de Dados

- Database único não é necessário mas é usual.
- Mudança de schema pode alterar todos os serviços.
- Ter um conjunto de componentes compartilhados por todos os serviços que representam o banco de dados é uma alternativa, mas não costuma ser bom, muita complexidade e acoplamento.



- *Figure 13-6. Using a single shared library for database entity objects*

- Melhor é ter pedaços de entidades compartilhadas, separadas logicamente.
  - Assim mudanças ficam contidas a um pedaço da aplicação.



- *Figure 13-7. Using multiple shared libraries for database entity objects*
- Mantendo os domínios bem definidos, quanto mais particionado, melhor.
- Ainda assim pode ser necessário ter componentes comum a todos (e.g. `common_entities_lib` no exemplo)

- Mudança nela é bom que seja restrita ao time de database.

Exemplo

- Reciclagem de eletrônicos.
  - Várias etapas entre orçamento, recepção do objeto, checagem de estado, etc.
- Cada etapa (domínio) pode ser um serviço diferente, deployado independentemente.
- Separação entre databases de operações internas e o de informações do usuário, comunicação entre eles unilateral.

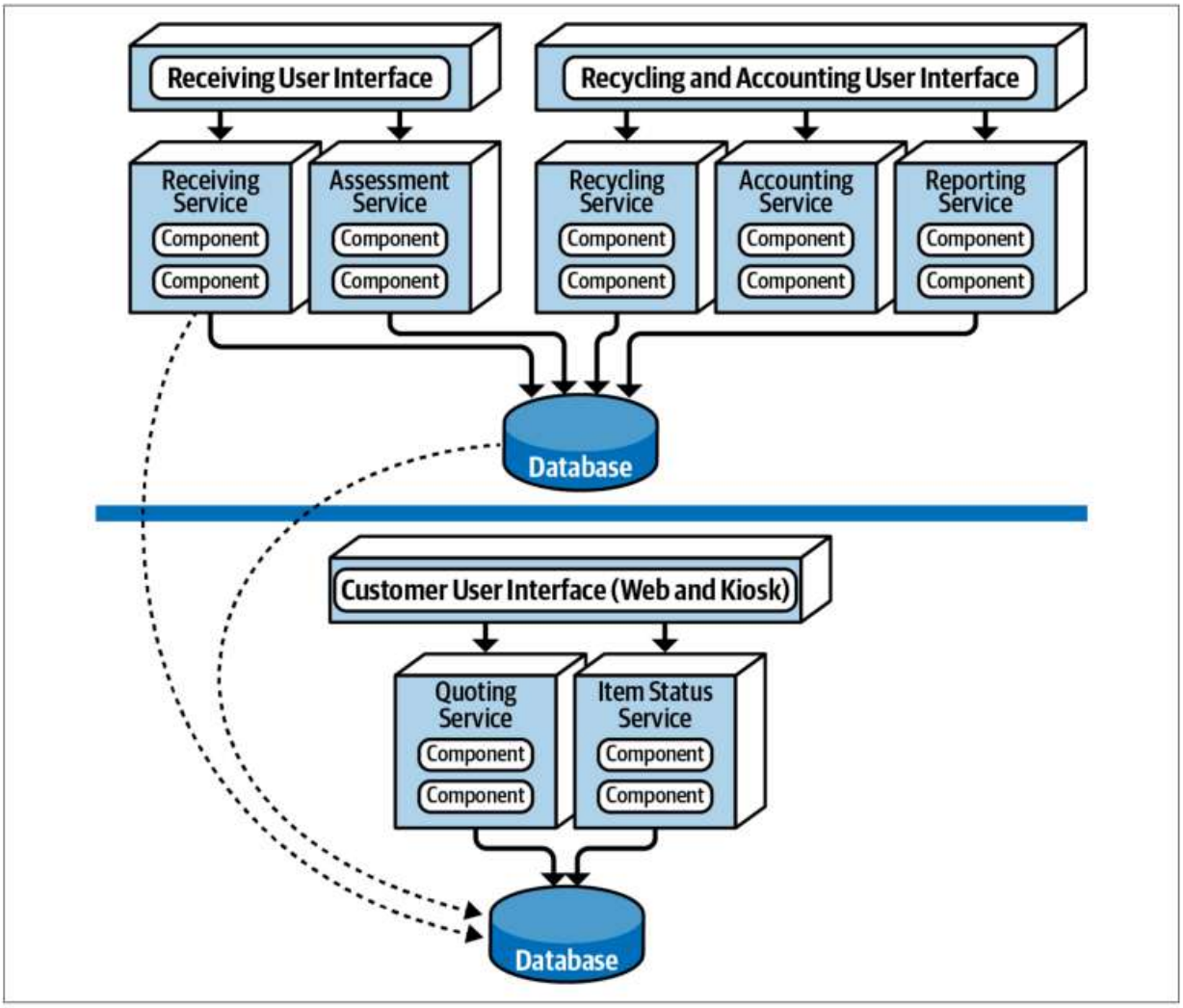


Figure 13-8. Electronics recycling example using service-based architecture

- Escalável, tolerante a falhas, segurança granular.
- Dois quanta de arquitetura (parte interna e a parte externa, cada um com um database)

Características

Architecture characteristic	Star rating
Partitioning type	Domain
Number of quanta	1 to many
Deployability	★★★★
Elasticity	★★
Evolutionary	★★★
Fault tolerance	★★★★
Modularity	★★★★
Overall cost	★★★★
Performance	★★★
Reliability	★★★★
Scalability	★★★
Simplicity	★★★
Testability	★★★★

- Particionada por domínio.
- Permite vários quanta de arquitetura.
- Bastante boa arquitetura em geral, sem ser excelente em nenhum atributo.
- Serviços são autocontidos e testáveis
- Escalabilidade não é ótima, já que serviços são grosseiros.
- Simples e custo-efetivo.
- Requer pouca banda e comunicação interna, com poucos pontos de falha

Quando Usar este Estilo de Arquitetura

- Quando requisitos de desempenho não são muito altos.
- Para Domain-Driver Design é um bom candidato.
- Quando preservar transações ACID é importante.
- Razoavelmente modular sem ser granular demais.
- Não requer tanta coordenação de muitos componentes. Exige alguma orquestração e coreografia.
    - **Orquestração:**

        - Coordenação de vários serviços por meio de um serviço mediador separado
    - **Coreografia:**
        - Coordenação de vários serviços pela forma com que os serviços se comunicam, sem um mediador.