Dynamic Reconfiguration of Evolving Web Services

Piotr Kaminski and Hausi A. Müller
Department of Computer Science
University of Victoria

Marin Litoiu
CAS Toronto
IBM Canada Ltd.

DEAS: Design and Evolution of Autonomic Application Software

- Sponsored by IBM Toronto and NSERC: $750K over 3 years
- Principal Investigators
  - Hausi A. Müller, University of Victoria
  - John Mylopoulos, University of Toronto/Trento
  - Marin Litoiu, IBM Canada Ltd.
- Over 15 PhD and MSc students involved

I

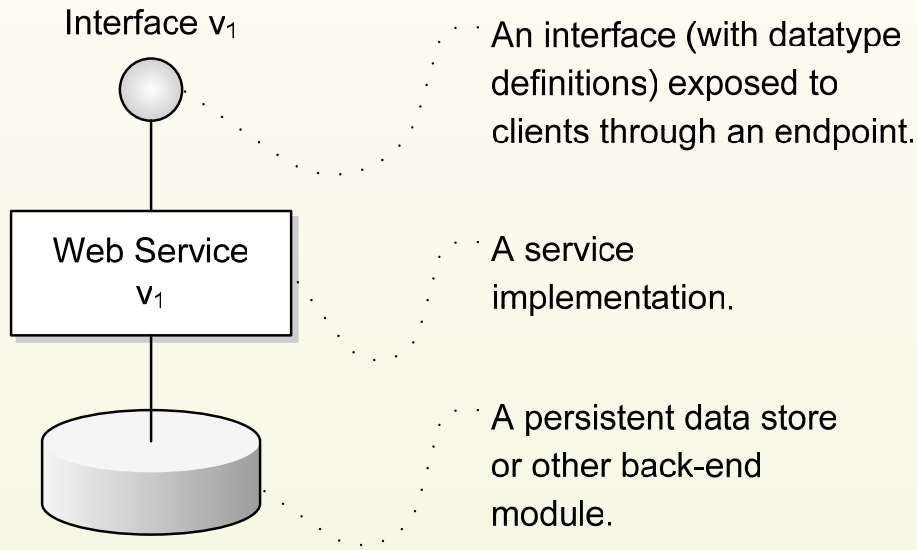
- Investigate methods for designing and evolving high-variability, self-managed systems using goal-driven requirements engineering methods
- Develop an analysis framework for AC application architectures using ABASs
- Investigate methods for evaluating complex tradeoffs
- Self-configuration of web and grid services
- Trust nomenclature for building AC systems incrementally

Goals and Results

- We are looking at:
  - autonomic self-configuration
  - ➔dynamic redeployment
  - ➔ ➔evolution management

  ... from a design-time perspective

  ... applied to web services



- Our results thus far:
  - The Chain of Adapters design technique for version management
  - An Eclipse/WTP (Web Tools Platform) plug-in to help apply Chain of Adapters to a WSDL/SOAP web service
  - Support from IBM Autonomic Computing and IBM Web Services VPs

The Challenge

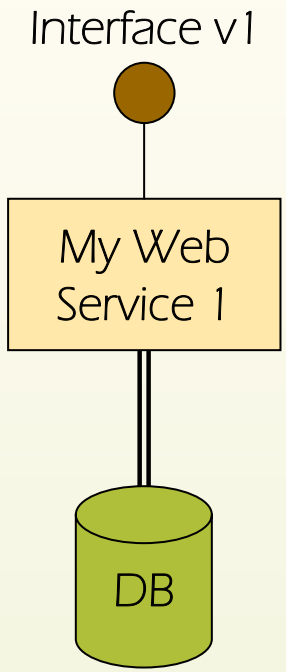
❶ Support backwards-compatible web service evolution

❷ Minimize cost of supporting older versions

❸ Simple for service developers, transparent for client developers

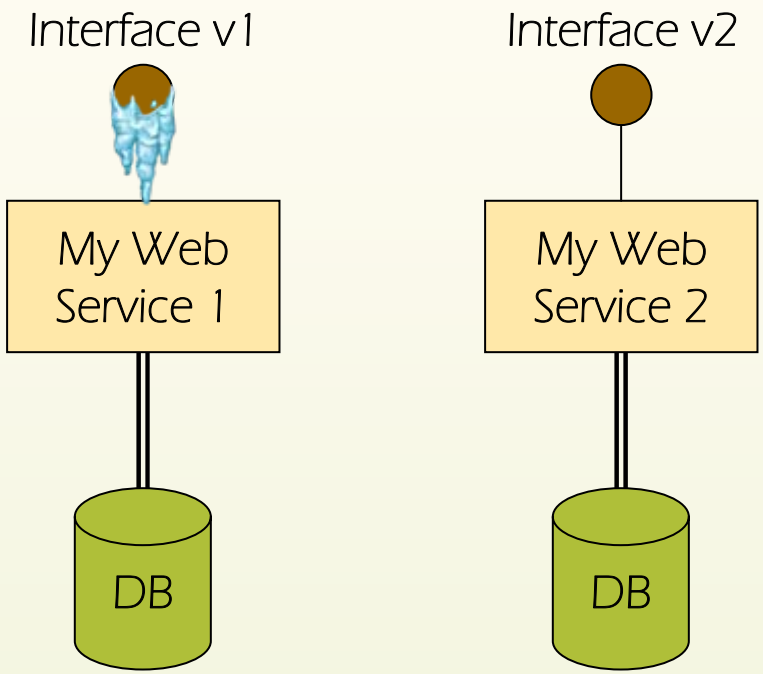
For human administrators as well as self-managing systems

Exhibit A: standalone versions



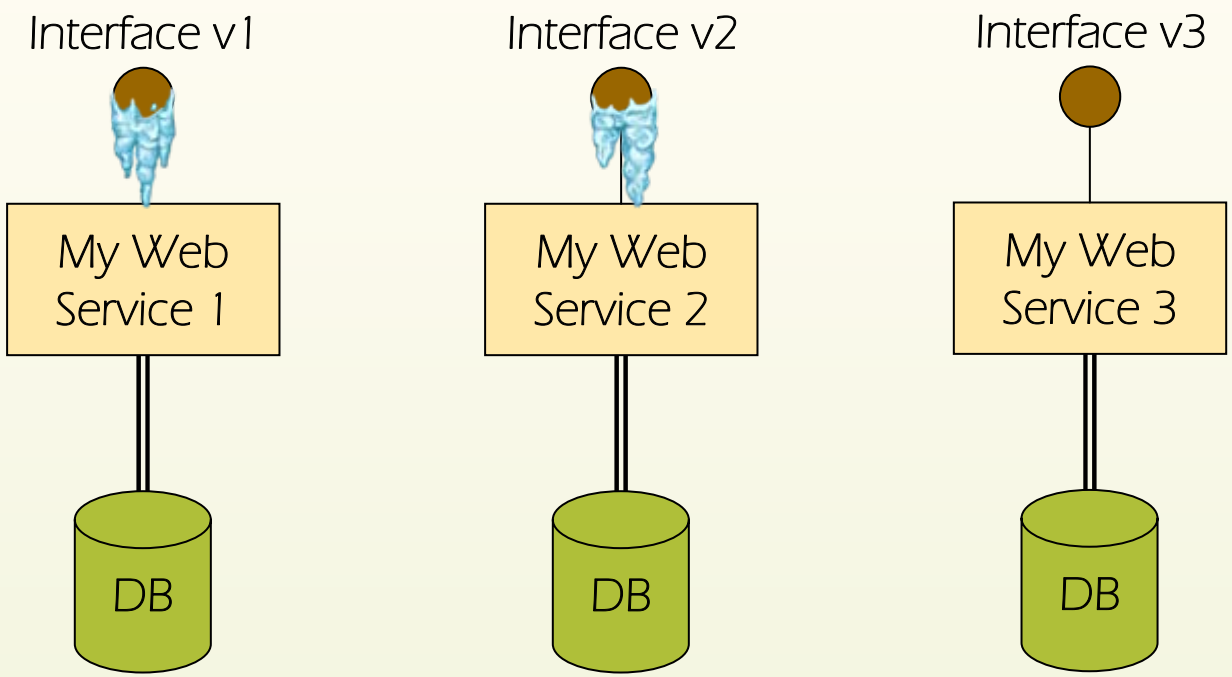
- ☑ Safest: old versions unaffected by new ones
- ☒ Maintenance, consistency and scalability headaches

Exhibit A: standalone versions



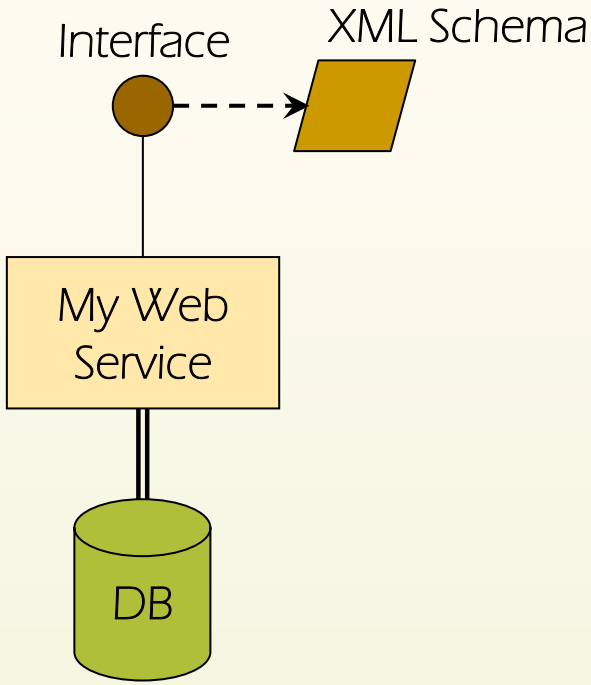
- ☑ Safest: old versions unaffected by new ones
- ☒ Maintenance, consistency and scalability headaches

Exhibit A: standalone versions



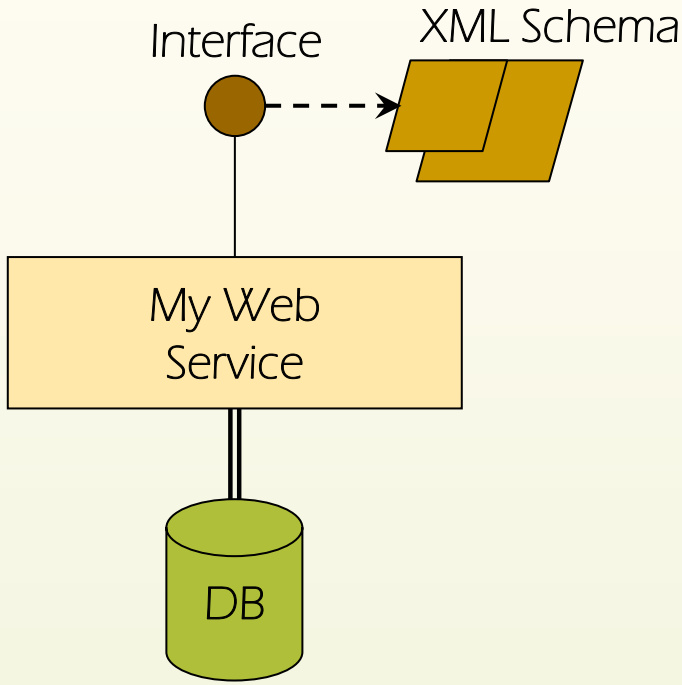
- ☑ Safest: old versions unaffected by new ones
- ☒ Maintenance, consistency and scalability headaches

Exhibit B: schema extension



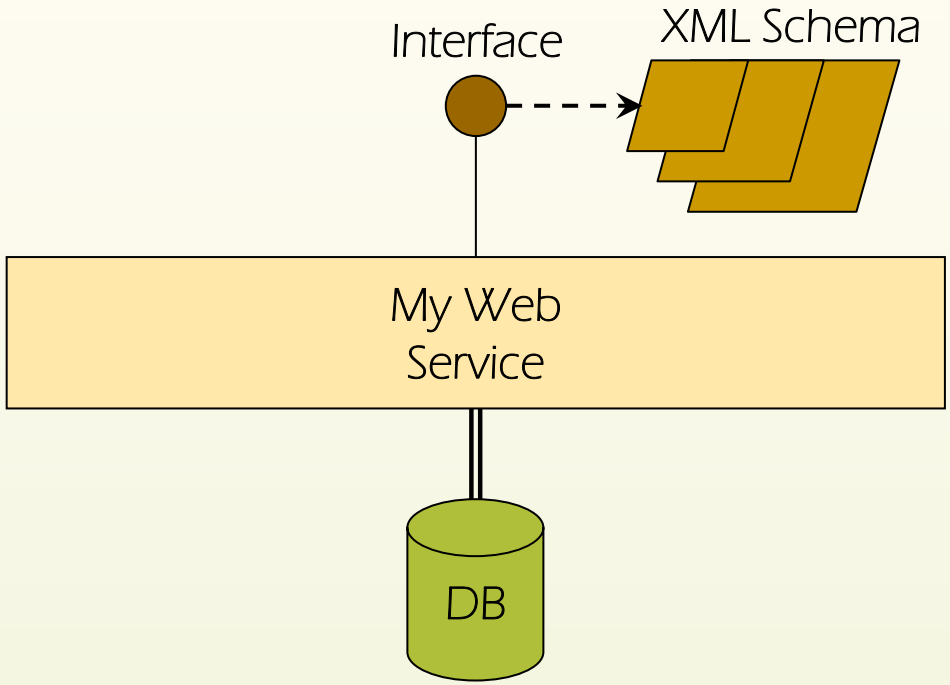
- ☑ Single access point forever, single codebase
- ☒ Tricky schemas, entangled code versions, constrained changes

Exhibit B: schema extension



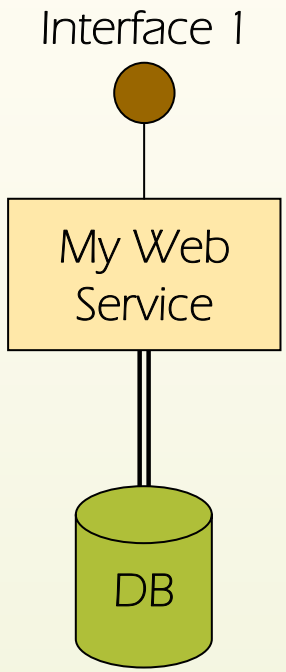
- ☑ Single access point forever, single codebase
- ☒ Tricky schemas, entangled code versions, constrained changes

Exhibit B: schema extension



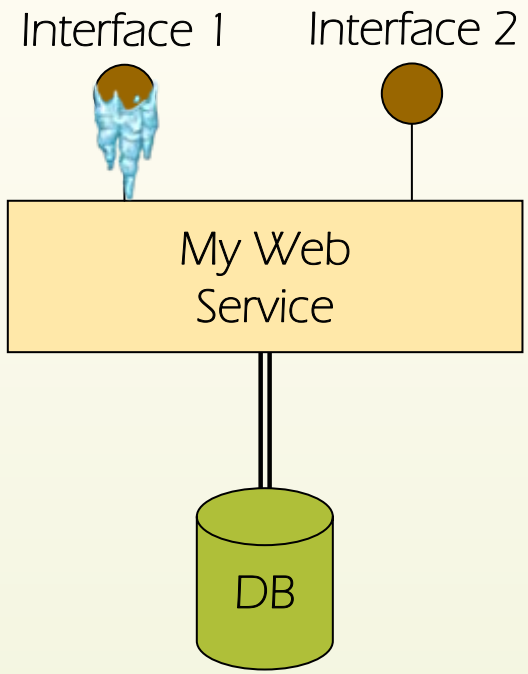
- ☑ Single access point forever, single codebase
- ☒ Tricky schemas, entangled code versions, constrained changes

Exhibit C: incremental interfaces



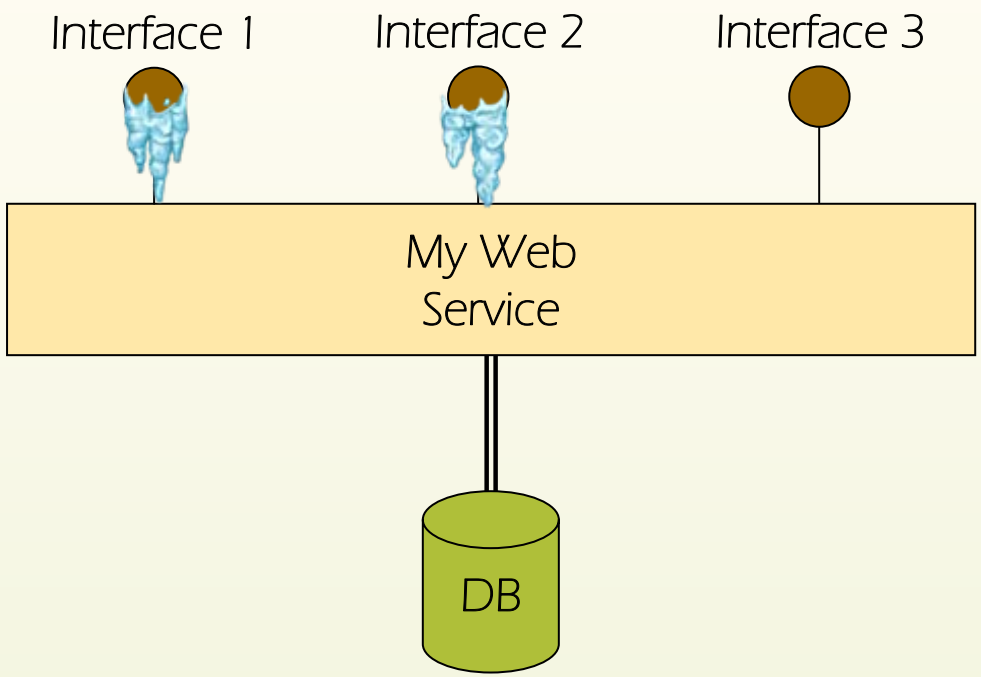
- ☑ Common and easy design, single codebase
- ☒ Scattered and bloated interfaces, entangled code versions

Exhibit C: incremental interfaces



- ☑ Common and easy design, single codebase
- ☒ Scattered and bloated interfaces, entangled code versions

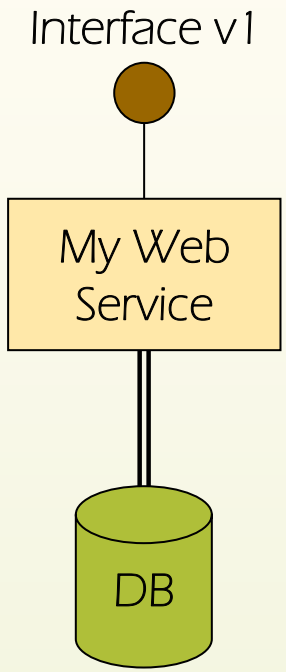
Exhibit C: incremental interfaces



- ☑ Common and easy design, single codebase
- ☒ Scattered and bloated interfaces, entangled code versions

Chain of Adapters

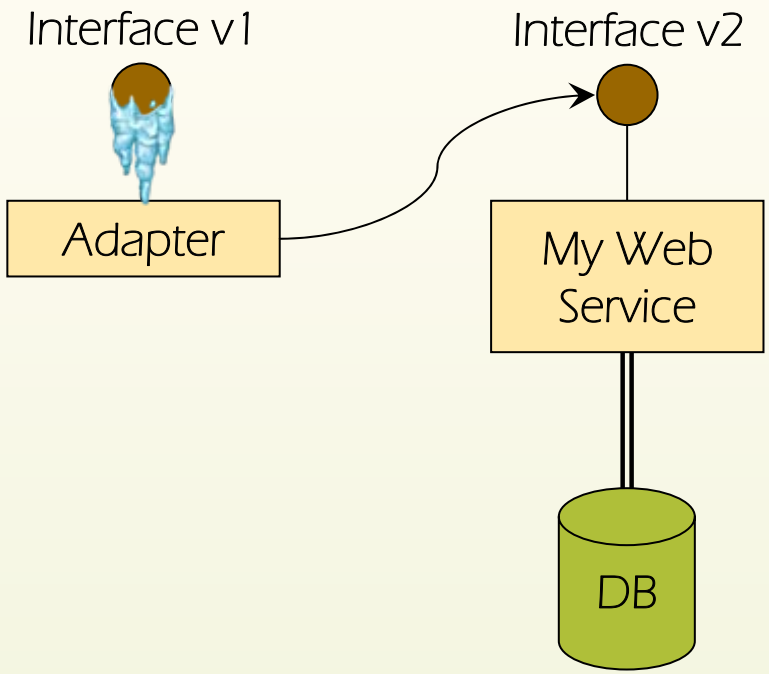
Also known in Haskell as ECT:
"**E**ternal **C**ompatibility in **T**heory"



- ☑ Versions separated, single code base, changes unconstrained
- ☒ Changes affect old versions,
  chain length impacts maintenance and performance

Chain of Adapters

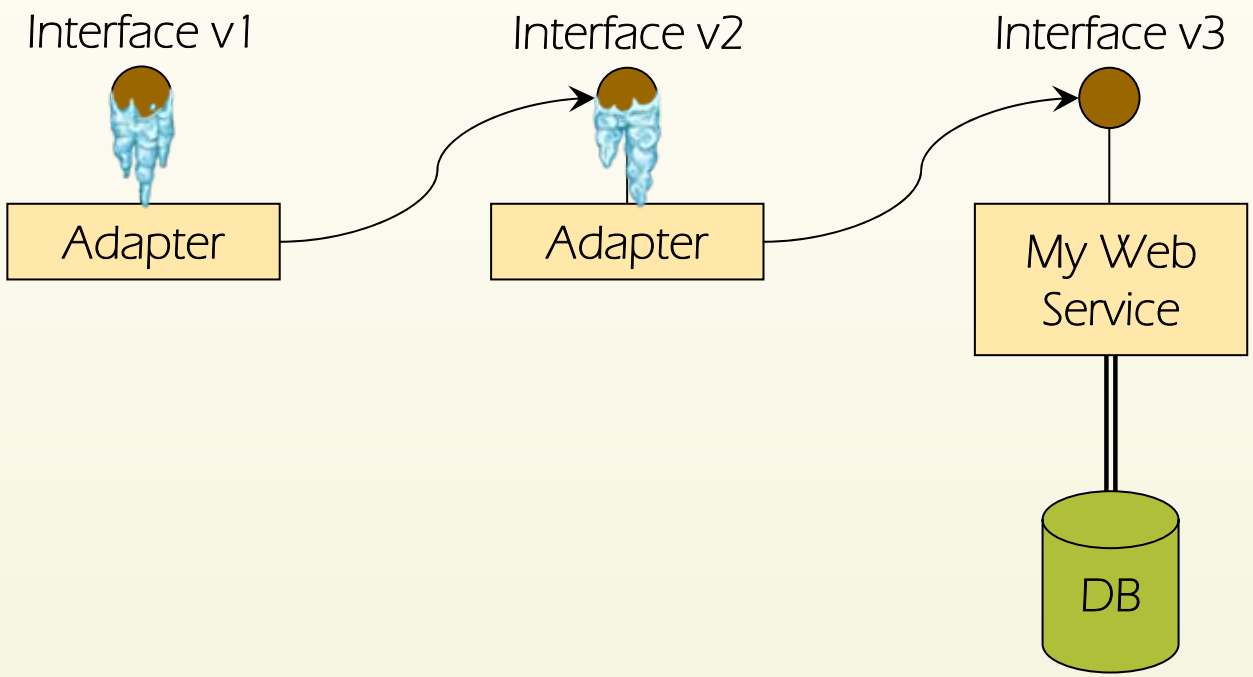
Also known in Haskell as ECT:
"**E**ternal **C**ompatibility in **T**heory"



- ☑ Versions separated, single code base, changes unconstrained
- ☒ Changes affect old versions,
  chain length impacts maintenance and performance

Chain of Adapters

Also known in Haskell as ECT:
"**E**ternal **C**ompatibility in **T**heory"

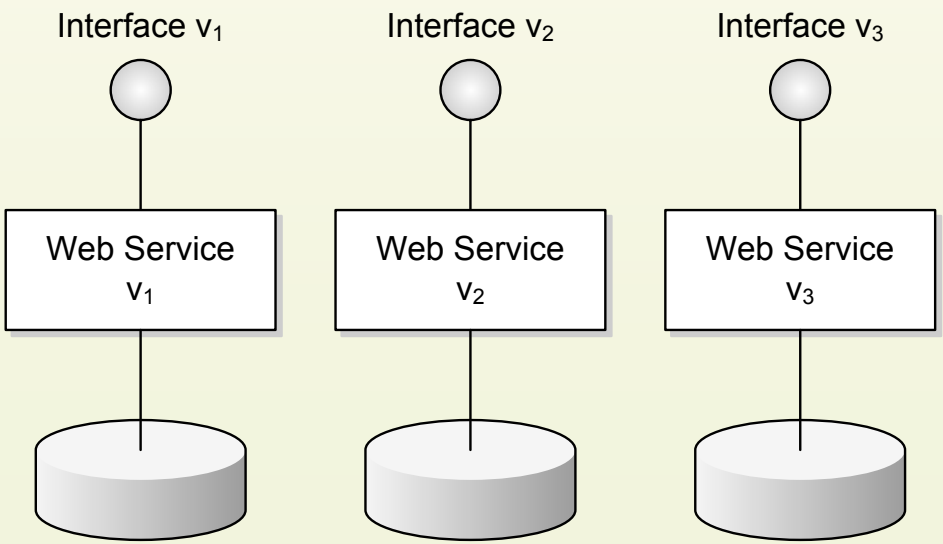


- ☑ Versions separated, single code base, changes unconstrained
- ☒ Changes affect old versions,
  chain length impacts maintenance and performance

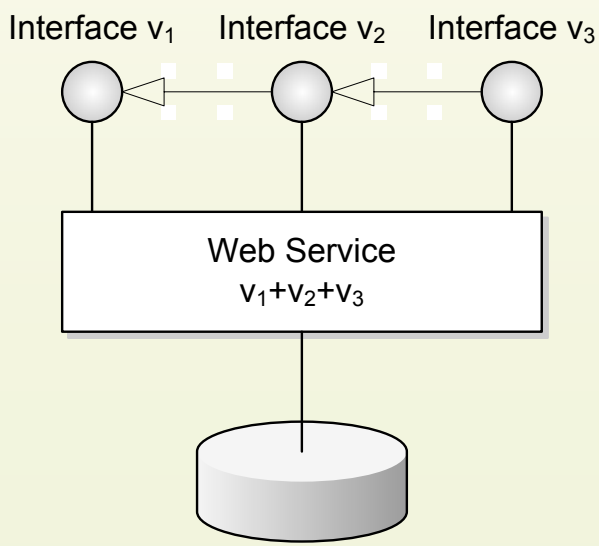
Requirements

1. Backwards compatibility
2. Common code base
3. Common data store
4. Untangled versions
5. Unconstrained evolution
6. Visible mechanism

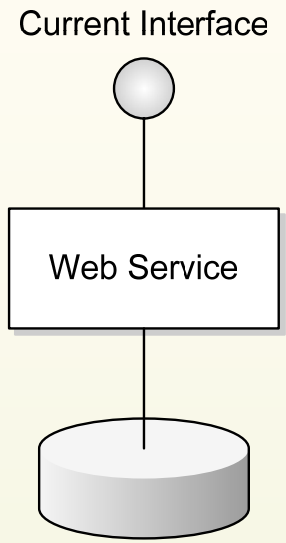
Isolated versions



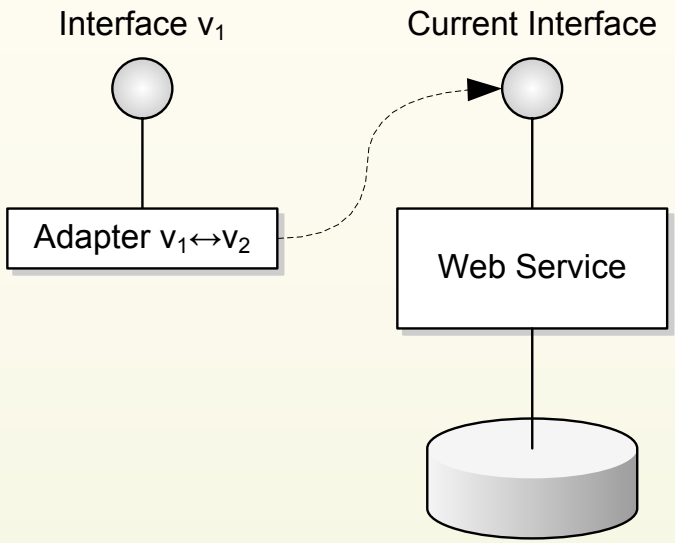
Tangled versions



Chain of Adapters (CofA)

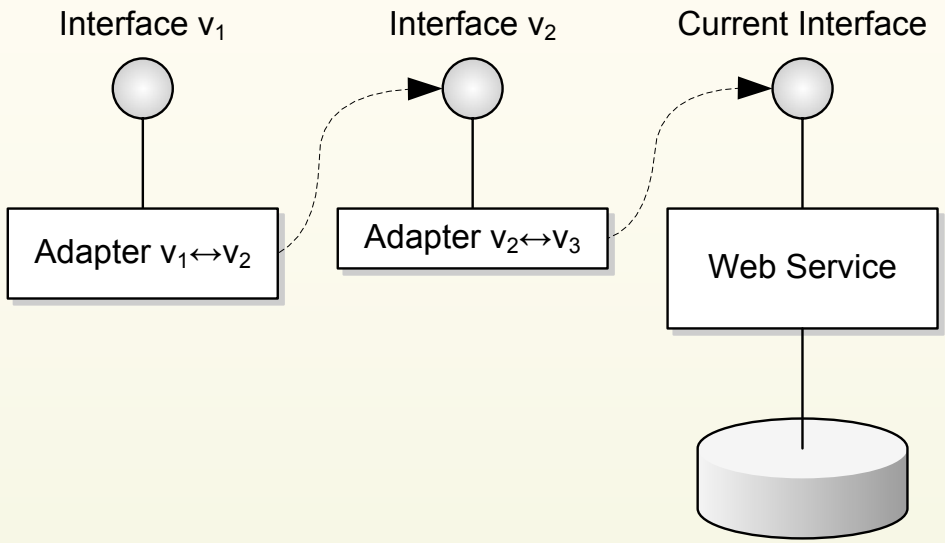


Chain of Adapters (CofA)



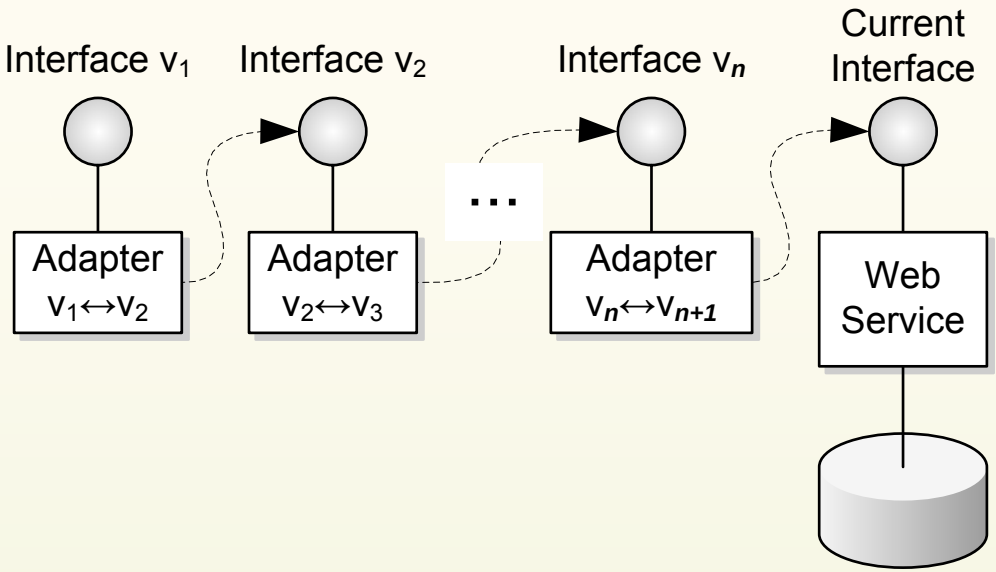
1. duplicate interface into new namespace
2. create trivial delegating adapter
3. publish frozen interface at new endpoint
4. make compensating changes in adapter

Chain of Adapters (CofA)



1. duplicate interface into new namespace
2. create trivial delegating adapter
3. retarget previous adapter
4. publish frozen interface at new endpoint
5. make compensating changes in adapter

Chain of Adapters (CofA)



Pros

- common code/data
- encapsulated versions
- transparent mechanism

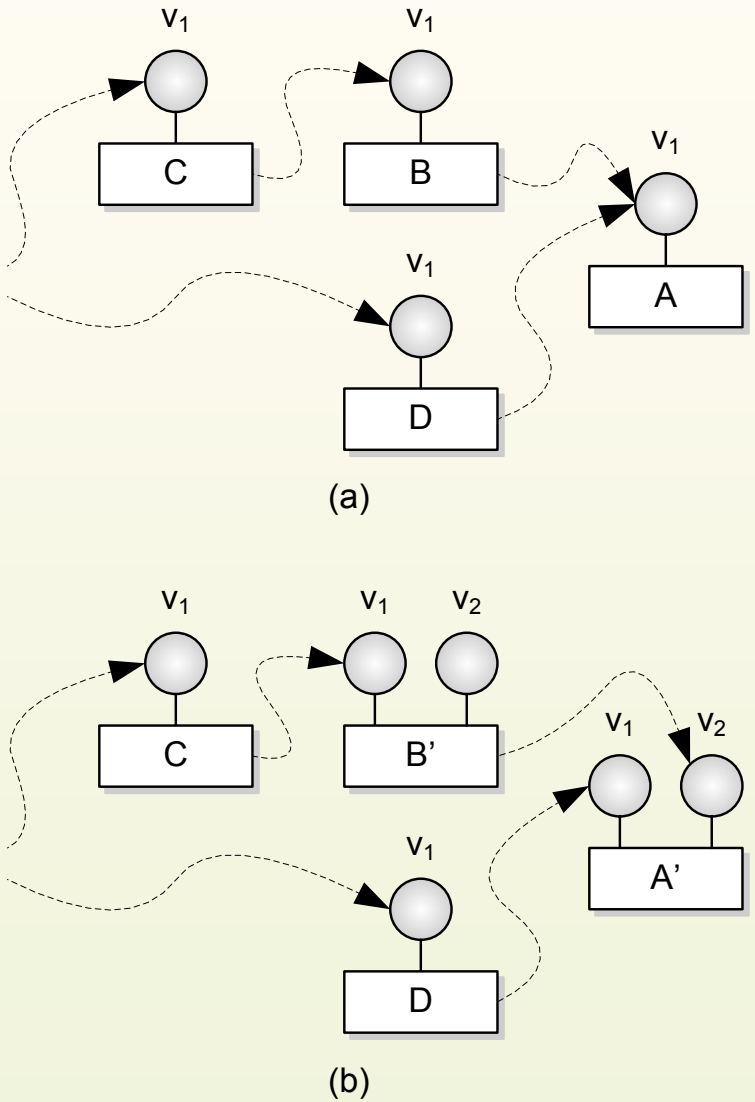
Cons

- backwards compatibility not guaranteed
- some constraints on evolution
- performance issues (manageable)

Reconfiguration Scenario

- Applying Chain of Adapters (CofA) *within* an application allows:
  - splitting the reconfiguration process into smaller chunks
  - shorter service discontinuities
  - easier failure recovery

(Each box in these diagrams encompasses a whole application, including its entire chain of adapters.)



Conclusions

- Present:
  - Design guidance for backwards-compatible web service evolution
  - Eclipse Web Tools Platform (WTP) “freeze & delegate” plug-in

- Next steps:
  - Rewrite Chain of Adapters plug-in for WTP 1.0/1.5
  - Adapt plug-in for IBM WebSphere Application Server (WAS)
  - Integrate plug-in into production WTP or other IBM tools
  - Investigate generalizing chain into tree
    - e.g. for bug fixes, or decoupled client/server development