Using AspectJ to eliminate tangling code in EAI-projects

Arno Schmidmeier

AspectSoft

AOSD-Conference

Boston 2003

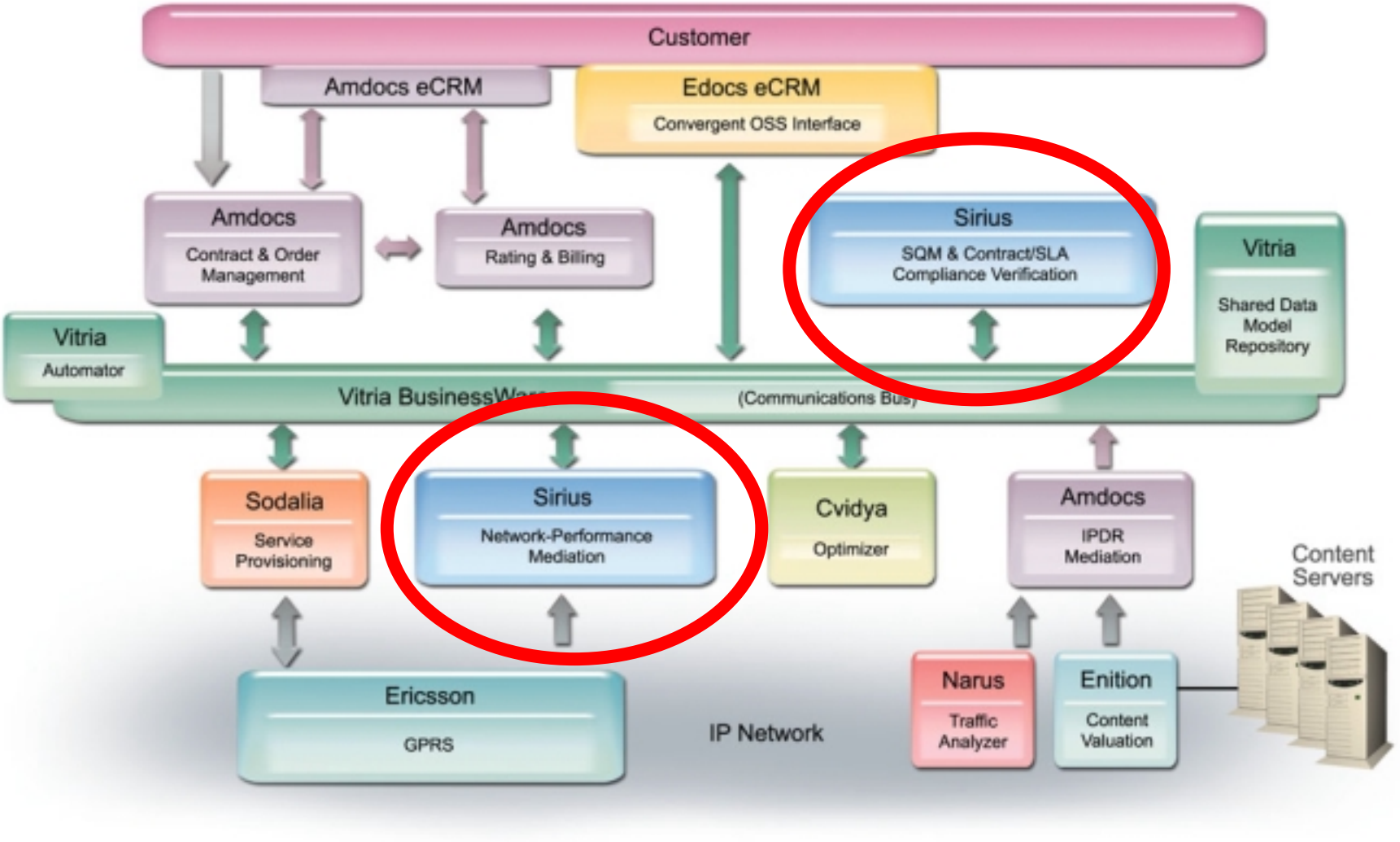
AOSD is about integrating software

- *how AO architectures are used to integrate Enterprise Software,*
- *where and why this approach creates currently a bunch of problems,*
- *how these problems have been overcome in several commercial projects by using AOP languages*

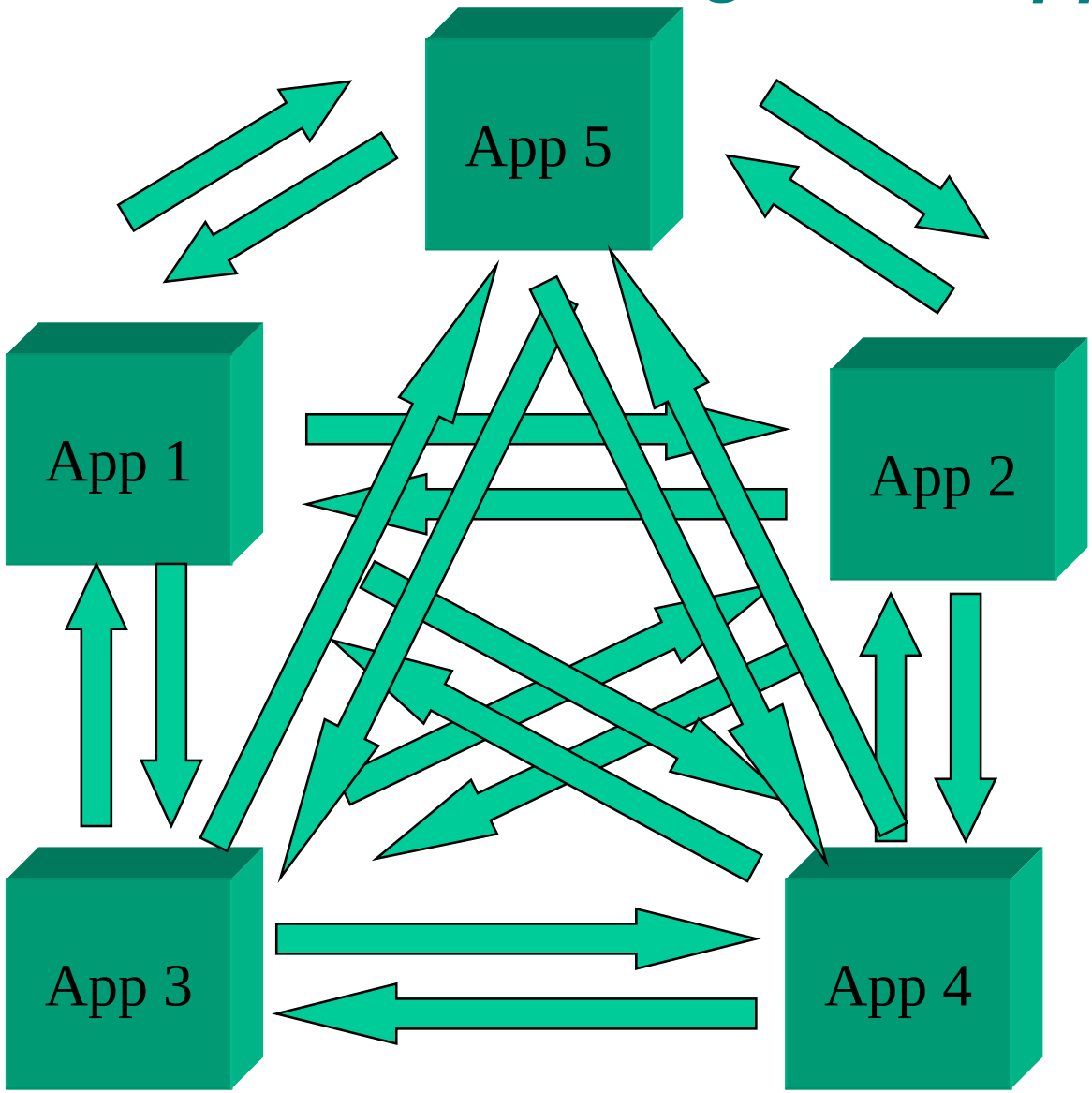
- *Note:*
  I report about that project from the viewpoint of an application vendor, who must integrate his product into an EAI scenario


VITRIA
Amdocs
cVidya
NETWORKS
T · · Systems · · ·
edocs
smarTrek
Wireless IP
SLA
Enition
swisscom
ERICSSON
SODALIA
SLM for Wireless IP
Lucent Technologies
Bell Labs Innovations
Sirius
simplicity out of complexity
NARUS

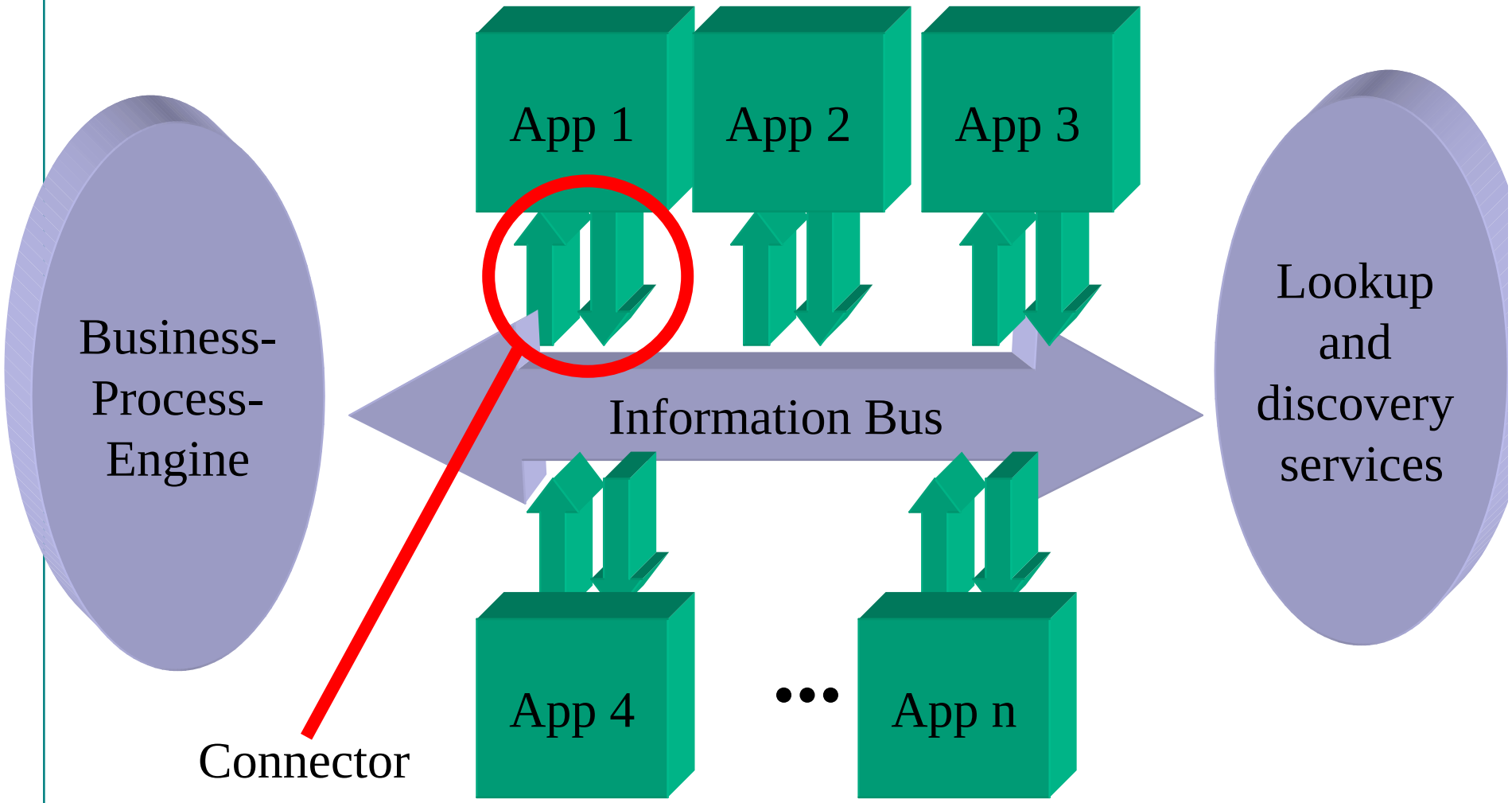
Our Jobs


Customer
Amdocs eCRM
Edocs eCRM
Convergent OSS Interface
Amdocs
Contract & Order Management
Amdocs
Rating & Billing
Sirius
SQM & Contract/SLA Compliance Verification
Vitria
Shared Data Model Repository
Vitria
Automator
Vitria BusinessWare
(Communications Bus)
Sodalia
Service Provisioning
Sirius
Network-Performance Mediation
Cvidya
Optimizer
Amdocs
IPDR Mediation
Content Servers
Ericsson
GPRS
IP Network
Narus
Traffic Analyzer
Enition
Content Valuation

Direct Integration Approach

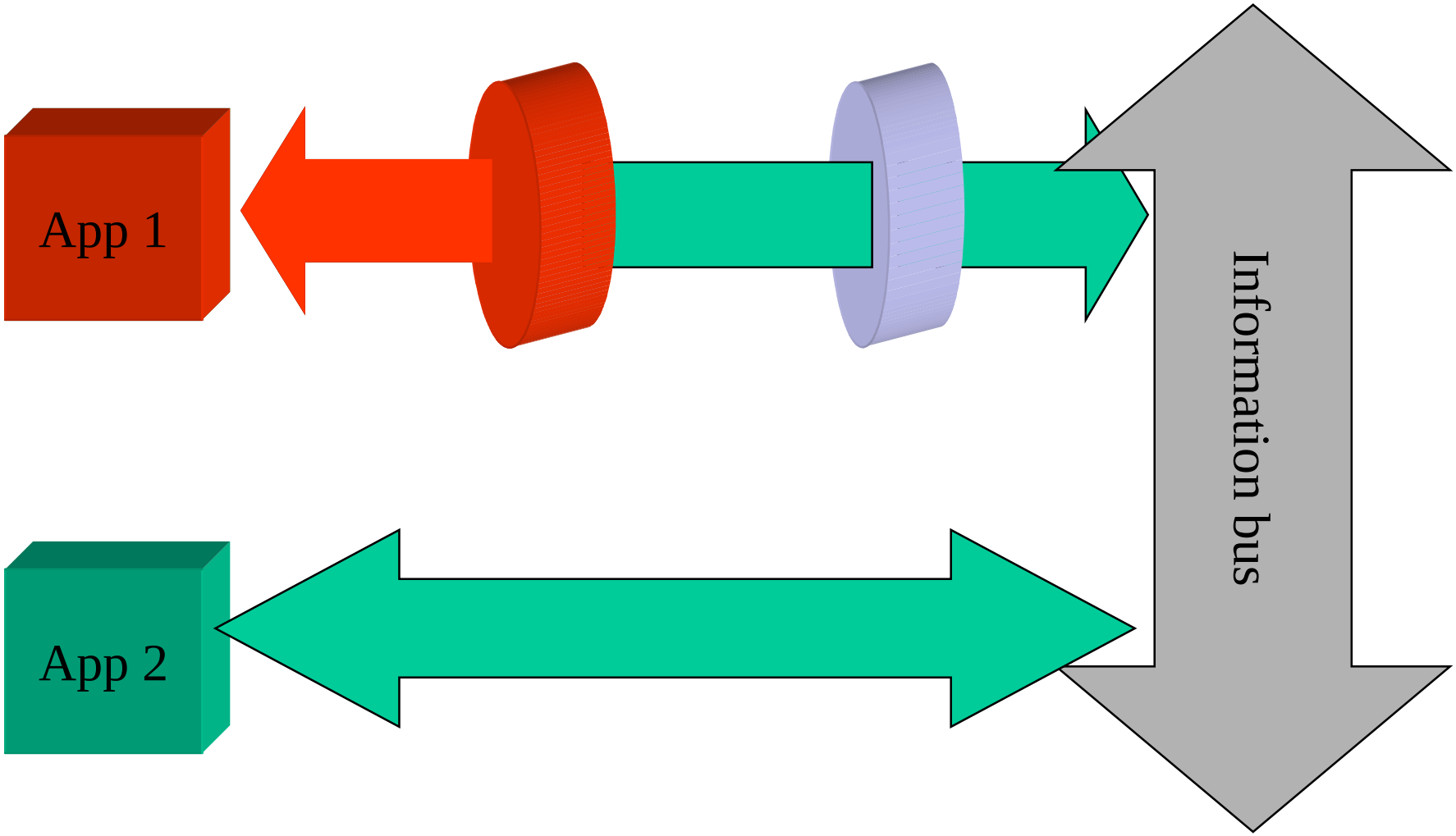


First improvement



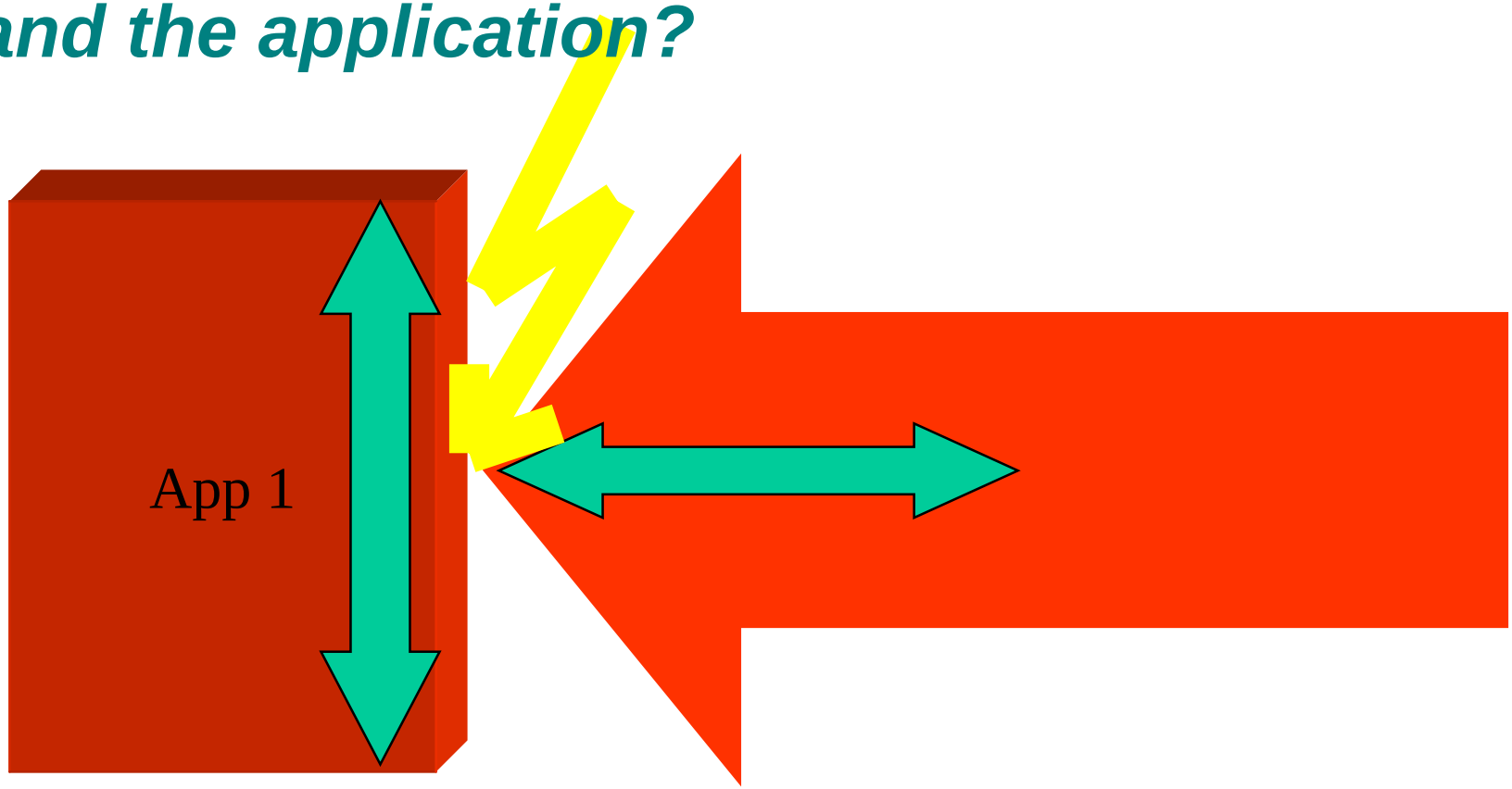
Evolution in the future

Using Composition filters



Logical Set-up

How to Integrate the connector and the application?



External Legacy Application API

- *Often:*
  - not available
  - Mismatch of
  - Too
  - implementa
- *D exposure the right funct*
- *Implementing some requirements w e to exp*

Not useable

As an application provider:
Use an internal API

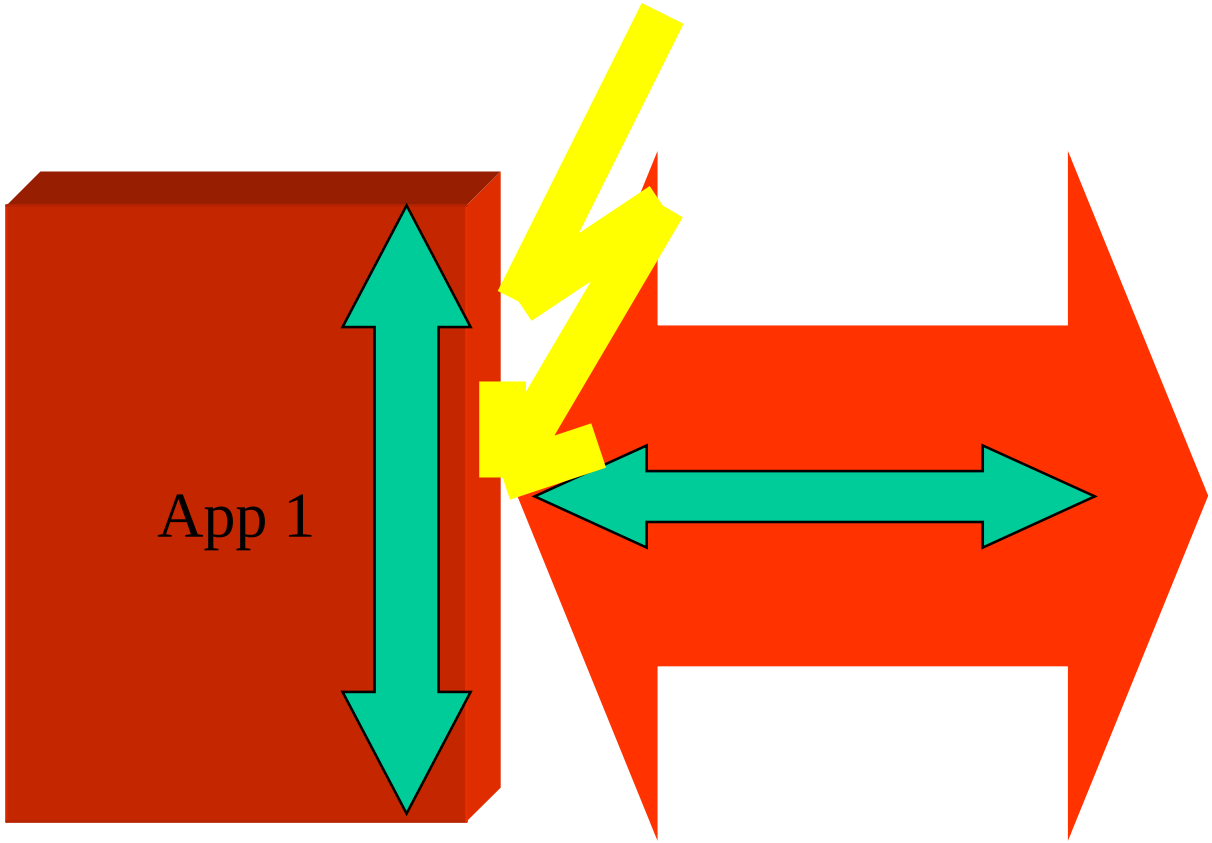
Using internal APIs

- *Using internal Java (AspectJ) APIs was fine*
  - We know the internal API
  - We could use all of our development tools
  - We had fine grained control
  - It was easy for us to spot the optimal places for the integration

However we decided :

- *not to change the Core code-files*
- *only add additional code*

Data-flow Mismatch



Impedance Mismatch

- *Pull-Scenario (insert, update, delete)*

  Solution: We used around-advice, which passed the information to the connector and called proceed

- *Internal information bus:*

  Solution: We registered the connector by the internal information bus, and created a virtual shared bus.

Impedance Mismatch

- *Push-Scenario*

  Observer-Pattern:

  We added a new observer instance, which forwarded to the connector

  Up call:

  We used an around advice, which passed the information over to the connector and called proceed

data stewardship

- *Data Stewardship: each objects is only managed by one application.*
- *On the bus: A common minimal required subset*
- *Problem: Often applications need more data than they store internally*
- *Solutions:*
  - We often piggy-backed data in introduced fields
  - Data stored in perthis, pertarget, or percflow aspects

Pre and post condition checking

- *A distributed Transaction-model was not possible*
  - A bunch of involved system can only work on a best effort policy
  - Transactions often do not make sense
  - Nobody would afford to implement it
- *Solution: Use of excessive Pre and post condition checking*
- *Must be implemented in the code*
  - Limitations of the EAI-Tools
  - Not all required information is sufficiently fast and easily available inside the connector or via public APIs
- *However: Simply and reusable done with AspectJ*

distributed Programming aspects

- *Caching*
- *Exception handling*
  - Retries and recovery
  - Fail over to a standby system
- *Tracing*
- *Auditing*
- *Examining delivery messages and retries*

For each method invoking the connector

plus

- *Heartbeat-checks*

Experiences

- *We missed some language features*
  - All are now implemented in AspectJ 1.1 except extensible pointcuts
- *Great Code reduction in the EAI parts ~95%*
- *EAI-Performance improvement (x2)*
- *Dramatic reduction of the downtime of the EAI-bridge*
- *We could place the aspects in a library and reuse it*
- *Security: Out of scope for these projects*
  Private secured networks are much easier

Experiences for AspectJ-projects

- *Having an experienced AO developer on board is a gem*
- *Capture idioms, potential patterns, anti-patterns, pitfalls, „how to do“s, etc. in a knowledge base*
  We used short powerpoint presentations.

Outlook&Conclusion

- *Poor quality of EAI-application is caused by the lack of aspects in many places*
- *Web-Services do not change the picture in general, they change (and improve) some details, but not the overall picture*
- *EAI-activities without aspects is doomed to fail*
- *EAI-architectures must make use from aspects*
- *We have to feed the knowledge of aspects in EAI-architecture bodies*
- *We need AO-middleware*