SERVICE ORIENTED ARCHITECTURE – THREE LEARNING EXPERIENCES

David Koenig
Michael Galarneau

May 2, 2006

Abstract

Service Oriented Architecture (SOA) offers companies a trifecta of more modular applications, reduced development costs, and faster time-to-market. Recent advances are making it easier-and-easier to implement SOA across a broad range of systems.

However, SOA is not a panacea. SOA initiatives often fail, not because of poor design or lack of technical capabilities, but because of failure to govern how services are actually built once the design is complete. Learning from less-than-ideal experiences is the first step toward gaining the benefits of SOA.

In this presentation, we will discuss three examples of SOA initiatives that failed to realize their lofty goals, and how we used these experiences to improve on later iterations of our SOA strategy. These examples span three different types of applications (a sales compensation system, a back-office administration system, and a real-time pricing engine) at three different companies. Each implementation achieved moderate success, yet under-delivered in ways that taught us where we needed to change our strategy and, more importantly, our expectations of Service Oriented Architecture.

© David Koenig 2006, SOA – Three Learning Experiences, May06

Getting Beyond the Hype

- **From the vendor point-of-view:**
  - ***"The combination of SOA and Web Services is very close to being the 'silver bullet' that companies have been looking for to:***
    1. ***Realize I/T's long-promised potential***
    2. ***Justify I/T expenses and capital outlays***
    3. ***Provide non-technical people a clear understanding of what I/T does, how they do it and their intrinsic value"***

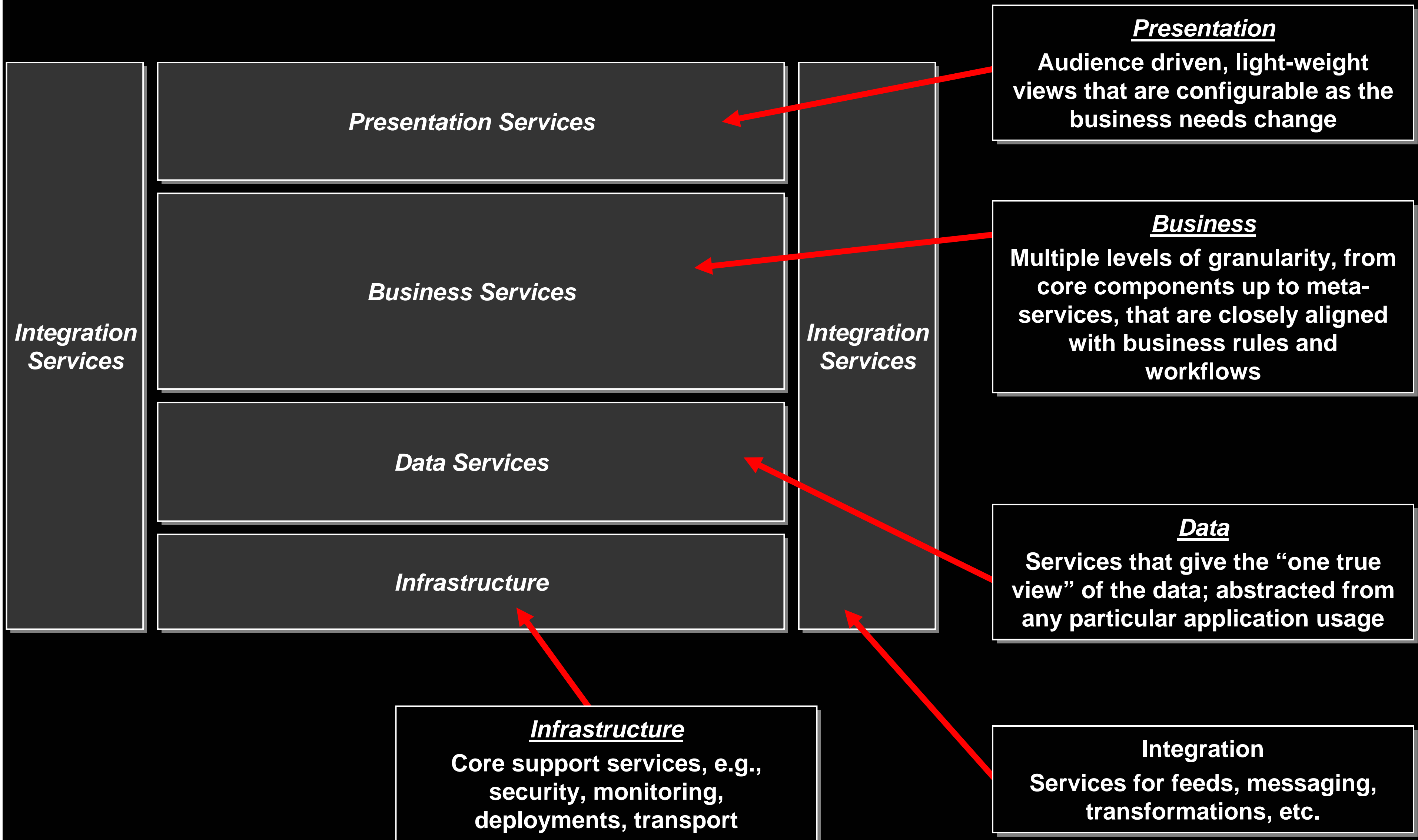
- **From the business point-of-view:**
  - ***"It always seems that this year's next big thing is just a way for I/T to get out of finishing last year's next big thing."***

- **What does SOA mean to a legacy I/T organization?**
  - **Constructing applications from components, services, and workflows**
  - **Design for interoperability and reusability**
  - **"Loosely-coupled" services and development teams (division of responsibilities)**
  - **Repurposing $100MM's of previous investment in existing, stable code**
  - **Strong standards and governance**

- **In other words, SOA is just good, basic n-tier development with focus on interoperability and standards**

© David Koenig 2006, SOA – Three Learning Experiences, May06

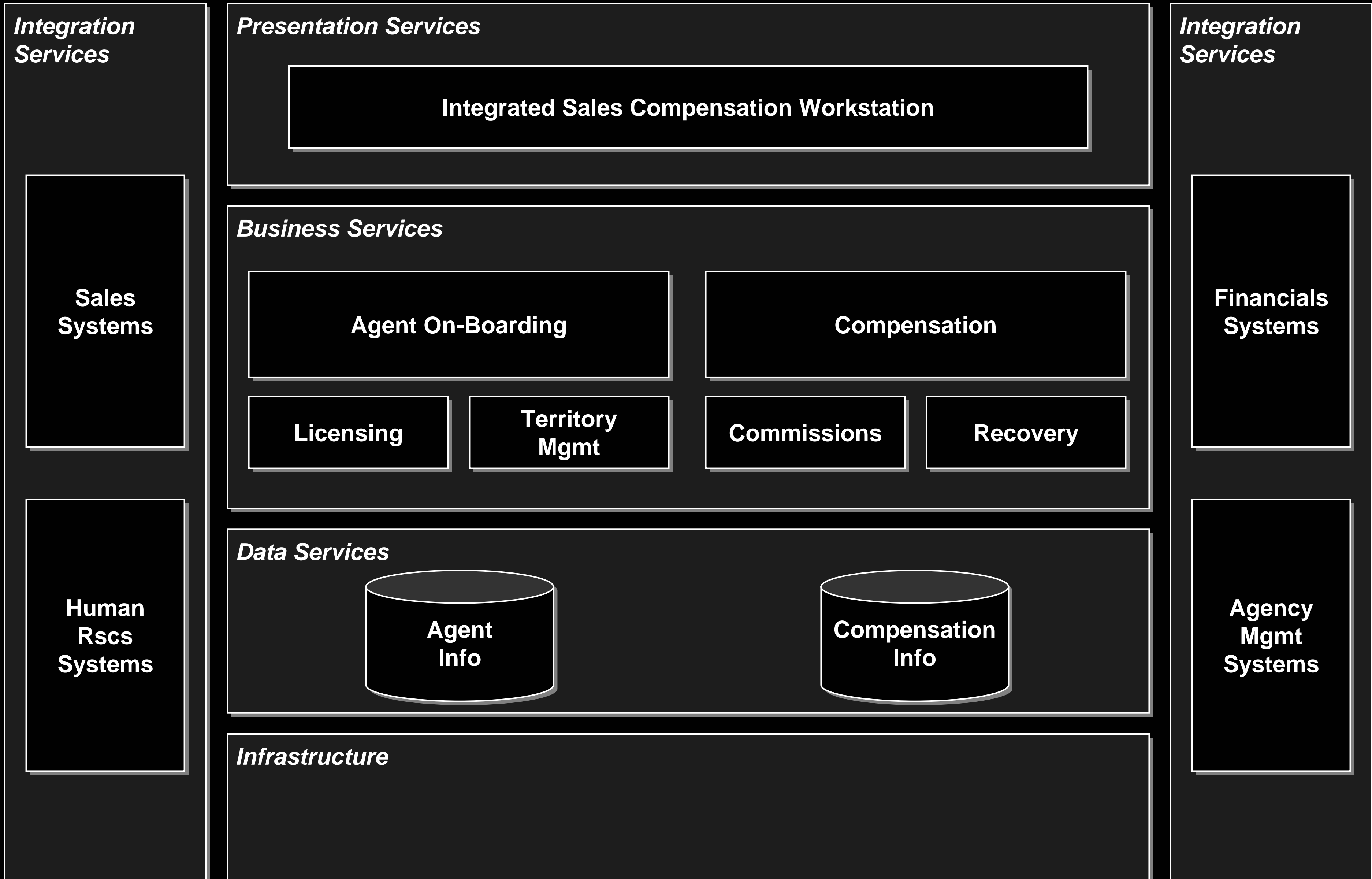
Framework For Deconstructing SOA Experiences



© David Koenig 2006, SOA – Three Learning Experiences, May06

Overview of the Three Learning Experiences

- **Different companies, different industries, different teams…same problems**
  - **Each experience is drawn from a composite of several projects**
  - **Management and lead architect same across projects**
  - **Each system is still active today and evolving as teams gain more experience**
- **Sales Compensation System: *"SOA As Implemented By Mainframe Folks"***
  - **Crude first implementation**
  - **Classic legacy "rip-and-replace" approach**
- **Back Office Administration System: *"Indigestion From Too Much SOA At Once"***
  - **Better understanding of component and service design**
  - **Alignment around business functionality**
- **Real-Time Pricing Engine: *"Right Architecture, Wrong Governance"***
  - **Limited only to functionality that warranted SOA**
  - **Integration and data flow core to service design**

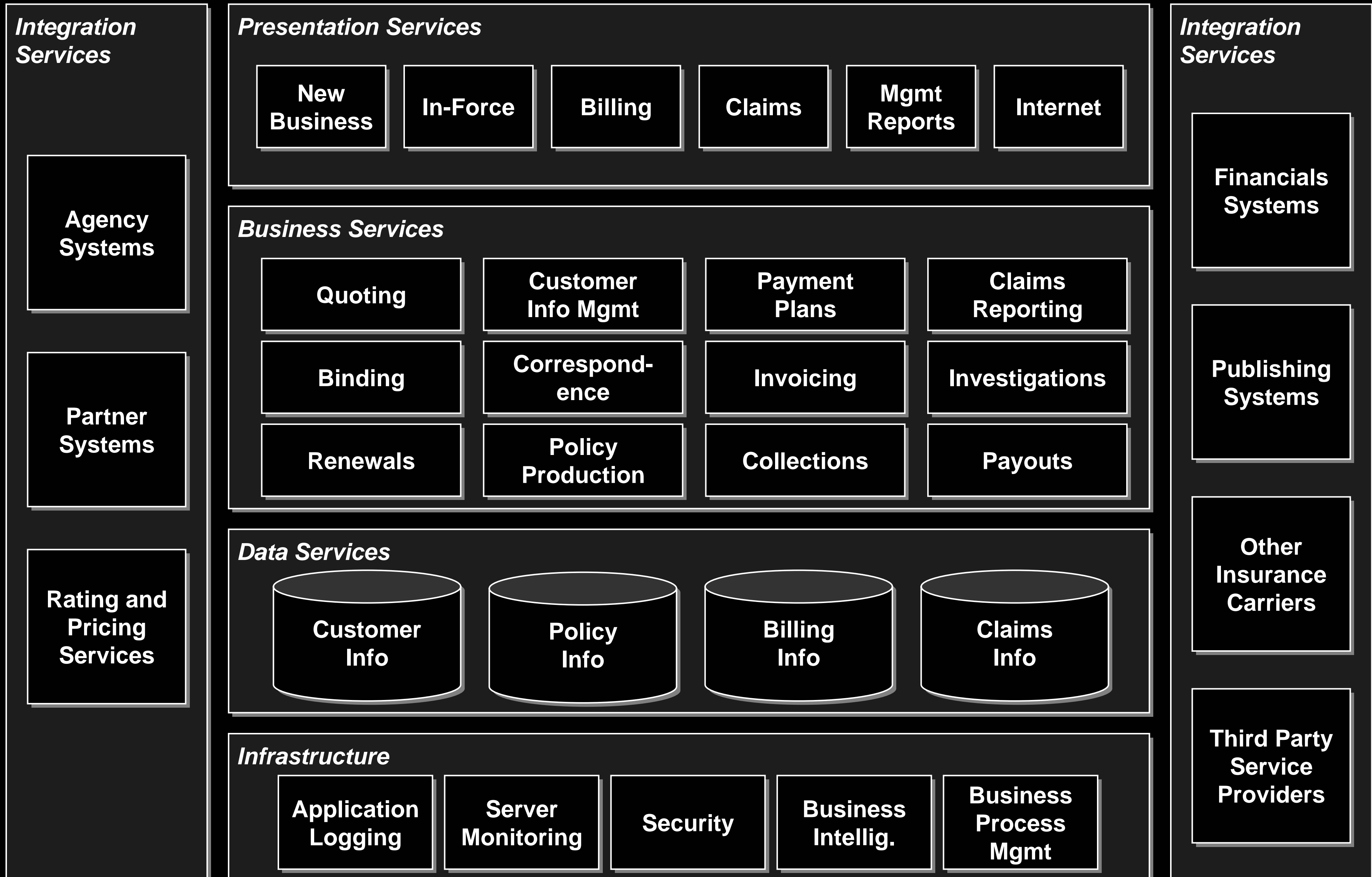


© David Koenig 2006, SOA – Three Learning Experiences, May06

Experience #1 – "SOA As Implemented By Mainframe Folks"

Services	*Good*	*Bad*
Presentation	• Well-defined business workflows led to quick user acceptance	• Classic fully-integrated, rigid interface • Heavy-weight client difficult to manage • Backlash from false configurability
Business	• Good modular design of <u>components</u> • Stayed faithful to J2EE standards	• Still a closed architecture; components embedded within monolithic app • Services not granular, not interoperable
Data	• Data services isolated from app-specific functionality	• Monolithic structure led to isolated data
Integration	• Designed for integration with up- and down-stream apps	• No ESB; relied on point-to-point and batch interfaces
Infrastructure	• Vendor independent; focus on Java standard and portability; not tied to O/S or app server	• Locked into legacy standards (e.g., CORBA); unable to migrate to newer standards as they were introduced

Experience #2 – Back Office Administration System

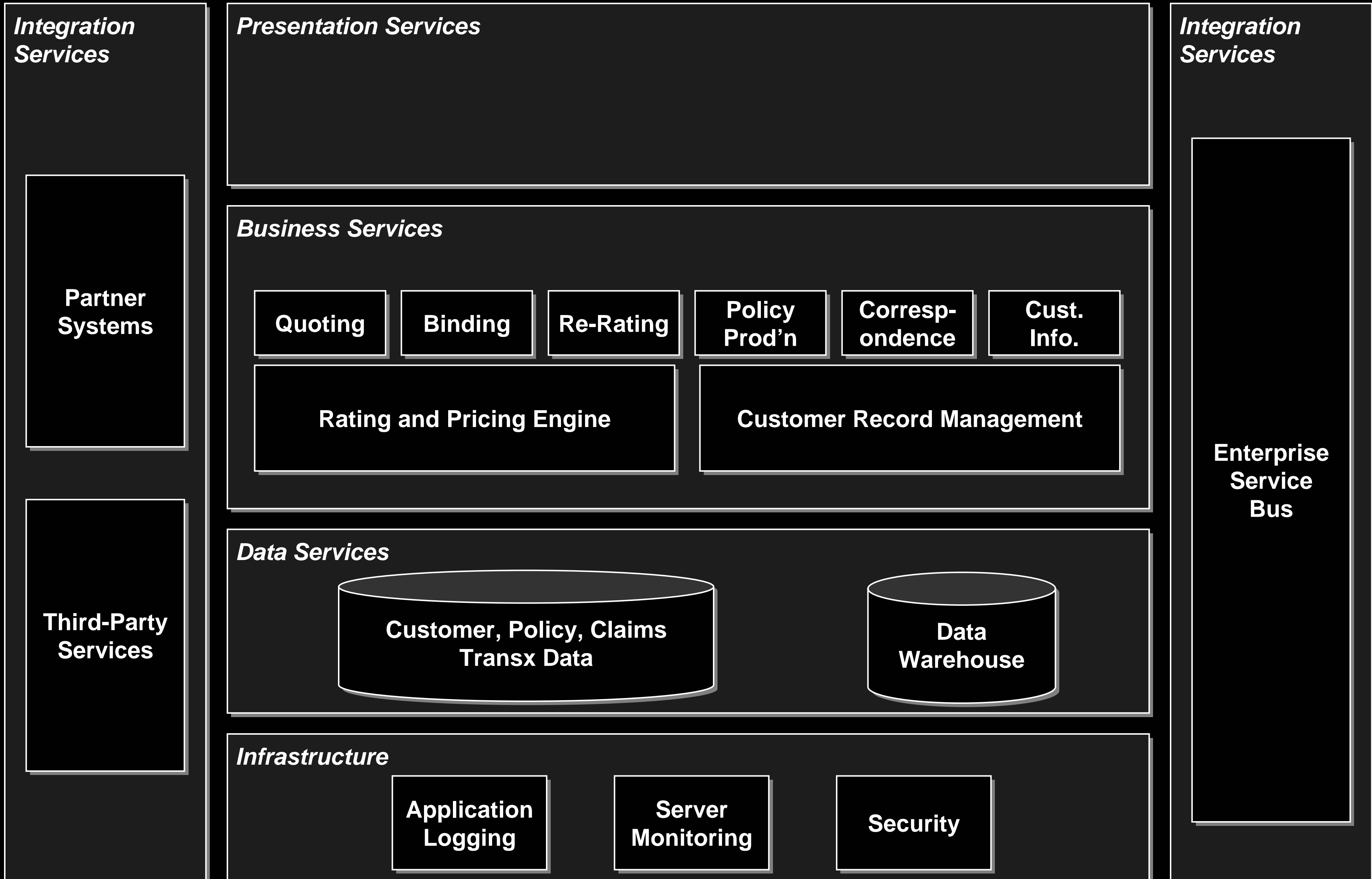


© David Koenig 2006, SOA – Three Learning Experiences, May06

Experience #2 – "Indigestion From Too Much SOA At Once"

Services	*Good*	*Bad*
Presentation	• Introduced portal capability, giving business visibility into services	• Home-brewed portal and content management framework • Late binding of loosely-defined services led to complex testing
Business	• Strong business alignment of components • Designed for maximum reuse • BPM tool for configuration of meta-services • Leveraged legacy mainframe code	• Too ambitious; too much SOA at once • Weak service governance, leading to sprawl • Services too granular, too loosely-define
Data	• Good metadata across many databases • Focus on efficient data access for services	• Fragmented data model difficult to manage • No thought given to integration with data warehouse
Integration	• MQ-based connectivity to host via message broker hub; easy to connect pieces	• Too many point-to-point interfaces • Non-standard message formats
Infrastructure	• Logging and tracking of business events • BI baked into the architecture	• Little security • Many basic services home-built (e.g., synch of databases)

Experience #3 – Real-Time Pricing Engine



Experience #3 – “Right Architecture, Wrong Governance”

Services	*Good*	*Bad*
Presentation	• No effort wasted on presentation for essentially back-end services	• No thought put into how to integrate with other presentation services
Business	• Strong business alignment of services • Right level of granularity • Mixed technology (C, Java, COBOL) with minimal rewrite	• No industry framework (e.g., open source, spring, struts) • Not designed for reuse (still app focused)
Data	• Single database for customer and policy data • Direct application access; well-tuned for transactions	• Overbuilt data model used by many apps; not abstracted from app tier • Not tuned for query or data mining • Large extracts to bolt-on data warehouse
Integration	• Designed to interact with variety of applications and front-ends • Tuned for speed (DTD-based)	• Home-brewed ESB • Too much app logic in the bus (so tempting for “time to market”)
Infrastructure	• Interfaces entirely XML • Fairly platform independent	• Wrote own security, monitoring, transformations

© David Koenig 2006, SOA – Three Learning Experiences, May06

What We Learned – Design Issues

- Didn't engage business early enough in design
  - Original services were SOA versions of current functionality vs. business-focused services
  - Business partners often think in terms of current systems vs. business processes
- Slow to embrace industry standards
  - Home-built technologies that weren't core to our businesses
  - Impossible to keep up with industry standards
  - Slow to implement core infrastructure services
- Had trouble defining right level of granularity for given platform
  - Too coarse…flexibility and time-to-market benefits were reduced
  - Too granular…responsibility for configuration fell on the business, leading to service sprawl and difficult testing scenarios
- Not everything had to be a service, some could have been just components
  - Exposed too much to the app layer; didn't develop enough workflow logic
- Too much focus on the service and not the underlying components
  - Should have spent more time on component design to get better behaved services
  - E.g., Save/Save-as

By failing to align services with business process, these initiatives didn't achieve desired level of reuse and interoperability

© David Koenig 2006, SOA – Three Learning Experiences, May06

What We Learned – Management Issues

- Introduced SOA in the wrong order
  - Start smaller, more evolutionary vs. “rip and replace”
- Abandoned governance when time-to-market became urgent
  - Doesn’t matter what approach you choose, but once you break from that approach, things will fall apart
- Failed to embrace “open source” management practices
  - All development and changes to be done by one central group
  - Didn’t define standards for other groups to follow
  - Should have been centralized governance, distributed development
- Allowed business functionality to trump architecture
  - Late binding services, combined with poorly defined interfaces, hurt ability to test all scenarios
  - Result was more flexibility but less robustness and reusability
- Didn’t realize that people want SOA to be more than what it is…but it is just basic n-tier architecture with ever-improving standards
  - Don’t sell the hype, it’ll only bite you in the end
  - If you can’t do basic n-tier development, you’re not ready for SOA

Jump in with both feet; there is no perfect project to start with; SOA is a discipline, not a killer app, and good enough is good enough

SERVICE ORIENTED ARCHITECTURE – THREE LEARNING EXPERIENCES

David Koenig
Michael Galarneau

May 2, 2006