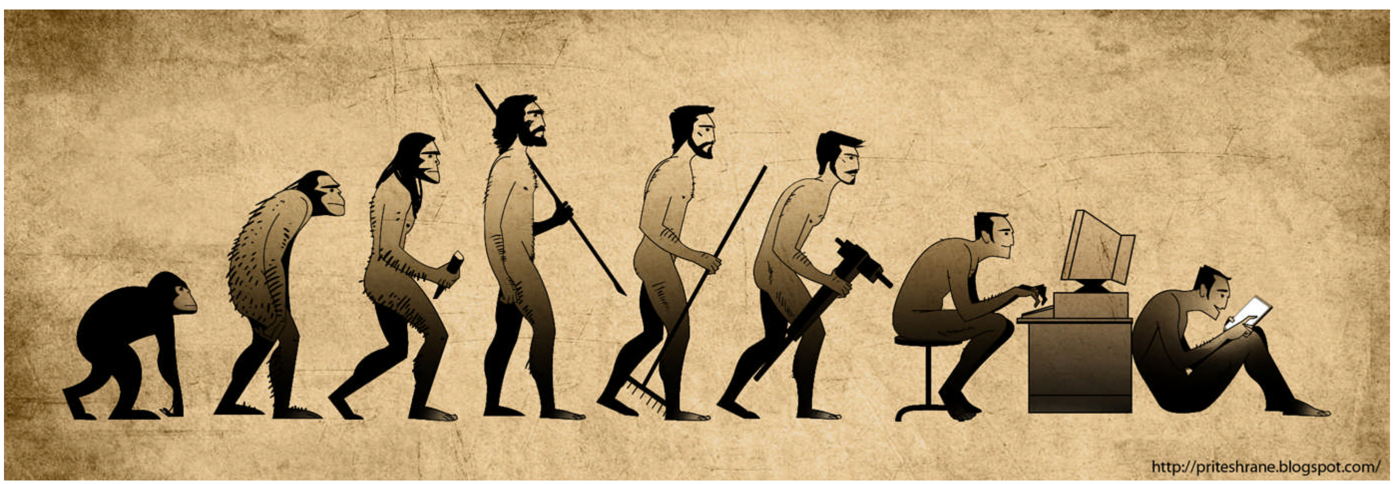
10 Years of Hype Cycles - Do We Forget Knowledge?

Aaron McConnell
Research Scientist
IU-ATC
School of Computing and Information Engineering
University of Ulster at Coleraine
Northern Ireland

Lost Technologies?

- Are existing technologies lost in the hype that surrounds a new one?
  - Yes, but isn't that the nature of change?

There is a cost involved in evolution



What causes hype?

- The hype around cloud is caused because the innovation is driven by business
- There is a clear drive to claim a stake of the new territory, which requires strong advertising campaigns and new products
- Innovation happens when capability catches up with demand

What have we lost?

Nothing?

- Technologies and methodologies can be resurrected if there is a new need for them
- As mentioned in a previous panel session: Neural Networks?

Objects, Components, Services at home and elsewhere (in the Cloud)

Claus Pahl
CloudCORE Research Centre – Deputy Director
IC4 National Cloud Technology Centre – Area Leader
Dublin City University
Ireland

CloudCORE

IC4
The Irish Centre for Cloud Computing & Commerce

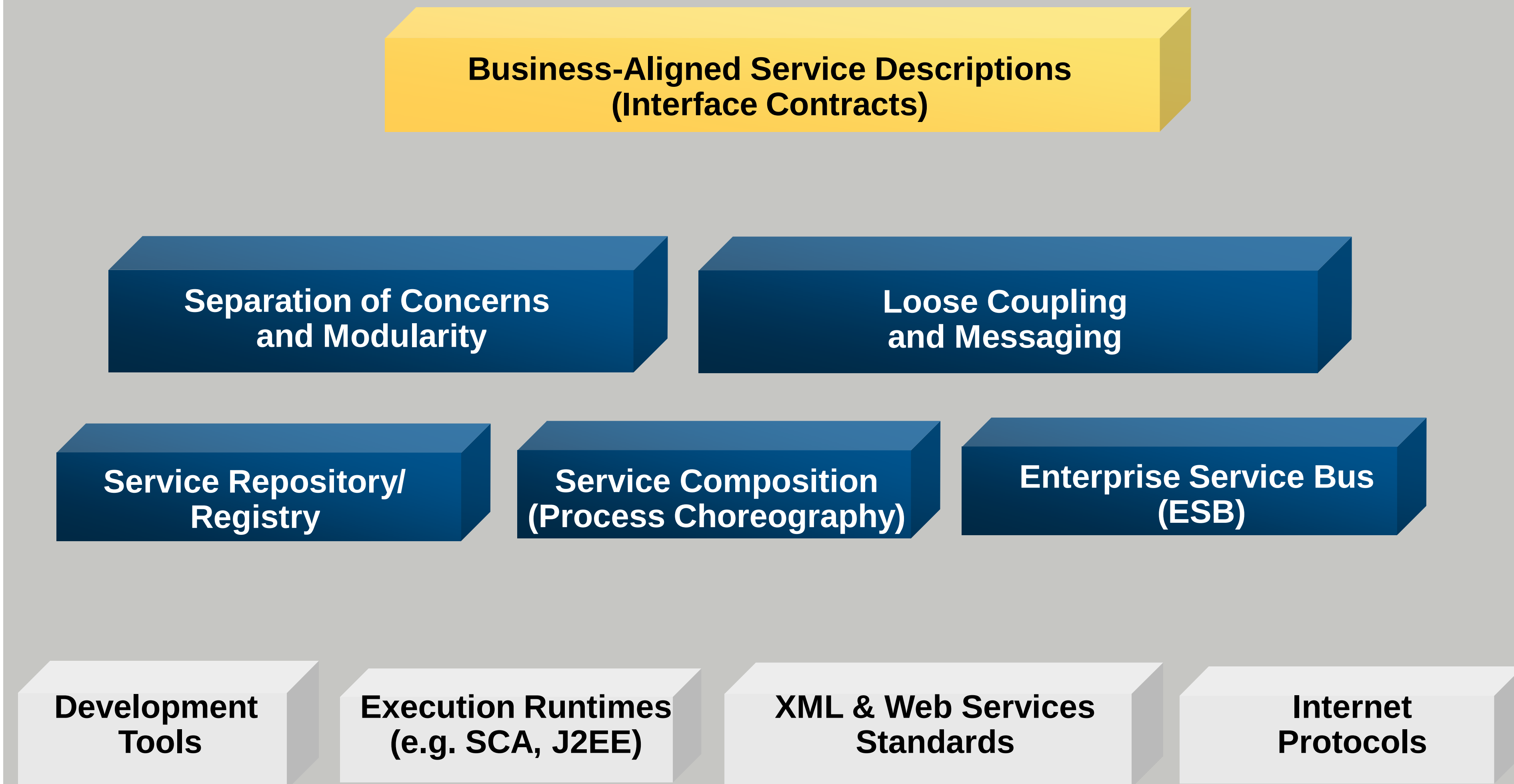
Architectural Styles

- *Principles* and *Patterns* define **Architectural Styles**.
- **Principles**: guiding software engineering principles linked to concrete benefits in terms of costs and quality (of software and processes)
- **Patterns**: solution templates that support the enactment of principles, achieving the anticipated benefits

Disruptive Development and Evolution

- *History* of *Software Development* and *Deployment Architectures*:
  - Object-Orientation -> modular software
  - Components and Middleware -> integrated software
  - Service-oriented Architecture -> distributed software
  - Cloud-based Architecture -> virtualised software
- Software Architecture Response:
  - *Mix & Match* of Patterns and Principles

SOA – Architectural Style



SOA principle: Modularity

- Motivation
  - Integrating monolithic applications is hard, e.g., traditional Enterprise Resource Planning packages
- Solution
  - Refactor to services, expose service interface only, hide implementation details -> *encapsulation*
- Consequences
  - Service *contracts* have to be defined and interpreted
  - Services have to be located and invoked in a *coordinated* manner
  - Service invocations have to be free of undesired side effects
- Roots and known uses
  - [Parnas], [Dijkstra] introduce modularization & separation of concerns
  - [Meyer] adds formal contracts with pre/postconditions and invariants
  - Component models such as CORBA, J2EE promote the concept

SOA principle: Layering

- Motivation
  - Service characteristics such as interface granularity and lifecycle vary: e.g., technical logging vs. claim checking business process
- Solution
  - Organize the SOA into 3+ *architectural layers*
- Consequences
  - More indirections, requiring communications infrastructure
  - Law of distribution: the best remote call is the one you don’t make
- Roots and known uses
  - Seven networking layers defined by [OSI]
  - Layers pattern originally described by Buschmann et al. in [POSA]
  - Patterns of Enterprise Application Architecture [Fowler]
  - e-business, on demand and web reference architectures

SOA principle: Loose coupling & messaging

- Motivation
  - Once applications have been modularized, dependencies between services occur
- Solution
  - Couple services loosely (several dimensions)
  - Messaging decouples in time, location, and language
- Consequences
  - Messaging means single implementation/endpoint by default (no remote objects)
  - Receiver is stateless per se, so conversational sessions require correlation logic
  - Asynchronous communication complicates systems management
- Roots and known uses
  - Enterprise application integration vendors have been promoting the concept for a long time;
  - Hohpe and Woolf define a pattern language for message-based integration

SOA Patterns: Enterprise Service Bus (ESB)

- A communications “architecture” that enables software applications that run
  - on different platforms and devices
  - written in different programming languages
  - use different programming models
  - require different data representations
- Foundation: well-established broker pattern [POSA]
  - Hub-and-spoke architecture known from EAI, i.e. many-to-many connectivity between loosely coupled parties – the ‚B‘ in ESB
  - Explicit, machine-readable service interface contracts – the ‚S‘ in ESB
  - Business alignment and high-end Quality of Service (QoS) – the ‚E‘ in ESB

SOA Patterns: Service composition

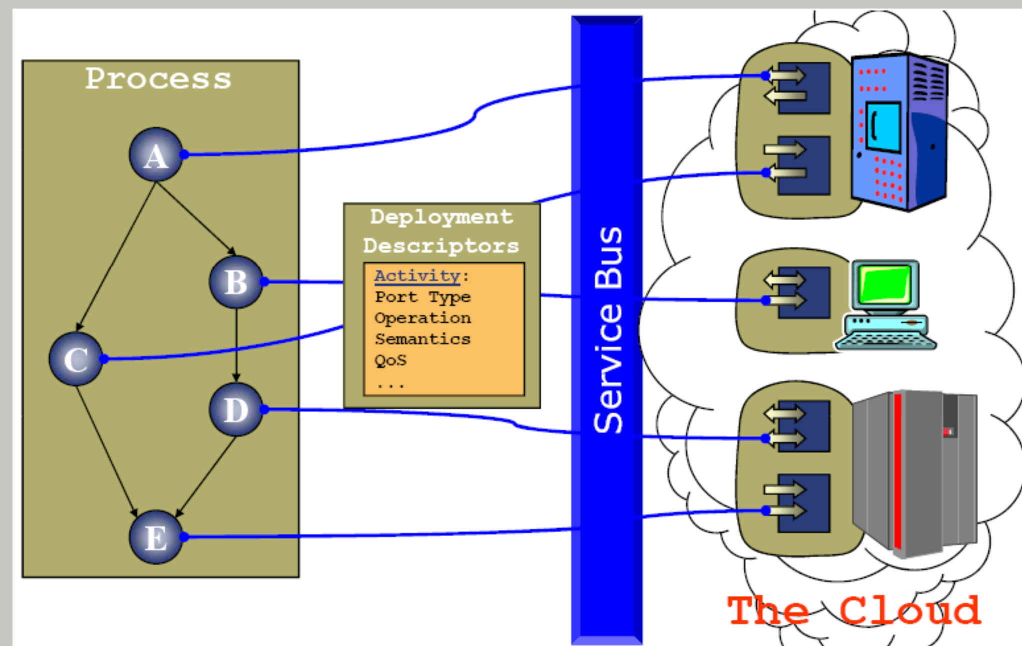
- Service composition
  - Choreography and orchestration mechanisms
  - Dividing process and atomic service layers
  - Programming model
- Foundations for process layer execution (workflows):
  - Business Process Management (BPM), Petri nets, Pi-calculus [Leymann]
  - One technology option is the Web Services Business Process Execution Language (WS-BPEL), standardized by [OASIS]

SOA Patterns: Service Registry

- Service registry
  - Build time service publishing and lookup
  - Runtime registration and lookup of service providers
  - Semantic annotations
  - Matchmaking
- Foundation:
  - SOA incarnation of naming and directory services
  - known from CORBA, J2EE, DCE, and other distributed computing technologies

Cloud – Architectural Style

- SOA + ???
  -> add Virtualisation as the principle !!!
- Concerns:
  - trust and privacy
  - precise semantics
  - QoS
  - multi-tenancy
  - provisioning
- Rooted in:
  - dynamic matchmaking
  - grid and utility computing
  - on demand computing



Cloud – Architectural Style

- Patterns:
  - Import SOA patterns:
    - Composition
    - Loose Coupling
  - Define additional Cloud patterns:
    - Marketplace – enhanced registry
    - Broker – negotiation to management lifecycle
    - …
    - Resource Migration – for elasticity, bursting, etc