Open Research Questions in Service-Oriented Computing, Enterprise Service Bus, Dynamic and Adaptive Service-Oriented Systems

Schahram Dustdar

Distributed Systems Group
Institute of Information Systems
TU Wien

http://www.infosys.tuwien.ac.at/Staff/sd/
Some papers to start with…


4 Evolutions – Implications

- Dependencies between parts of systems are no longer fixed and predetermined
  - Human team activities and team forms
  - Software services
  - Interactions between teams and services on infrastructures

- Ability to deal with context changes and unanticipated events and people
  - self-* behaviors:
    - e.g., self-adapting, self-organizing

- Support active objects providing service, such as
  - taggable objects (e.g., RFID, NFC)
  - artifacts
  - Sensor composition and sensor networks
Complexity, Interaction, Autonomy

- Heterogeneous systems increasingly connected
  - Integration becomes more complex

- Software – and Hardware-Architects cannot plan for all potential interactions upfront
  - Increased interaction dynamics of systems and processes (people and software services)

- Autistic software vs. Autonomic software

- Monitoring and Management of Internet-scale infrastructures becomes paramount -> shift to runtime
  - Autonomic & Services Computing including e.g.,:
    - Self-Healing
    - Self-Configuring
    - Self-Optimizing
    - Self-Protecting
    - ...
    - Self-* properties
Mastering Complexity

- Infrastructure/Software/Process/Teamwork Evolution

**Top-down approach:**
Process-driven and MDD approaches to master complexity and enterprise-scale change:
- model/build once -> use many times by many service consumers)

**Bottom-up approach:**
User-driven composition, Mashup approach for small-scale:
- build once -> use once)
Agenda

- Service Oriented Computing Research Roadmap
- Theme#1: Service Foundations
- Theme#2: Service Composition/Assemblies
- Theme#3: Service Management
- Theme#4: Service Engineering
- Summary
Extended SOA (xSOA)
Research Challenges

- **Methods and Models for Design, Development, and Evolution**
  - Service Composition for open complex and dynamic systems
  - Model-driven development for compliant SOC Systems
  - Context-based service composition

- **Service Discovery in Registries**
  - Dynamic Binding in Registries
  - Retrieval languages for humans and software services
  - Transient service providers and registries
  - Central/decentral/hybrid architectures

- **Protocols for Coordination and Interaction**
  - Coordination of dynamic services
  - Analyses of Service-Interactions (Mining, Patterns, Algorithms for optimization)
  - Interaction models and protocols (e.g., Policies, etc.)

- **QoS-optimized Internet-scale Processes**
  - Performance monitoring & management of service compositions and networked open dynamic systems
  - Metrics (e.g., Team forms, Orchestration Complexity, Coupling metrics)

---

Research Themes

- **Service Foundations**: runtime infrastructure, architectures, e.g., Enterprise Service Bus, modes of service delivery = mobile, palm tops, hand held devices, networks.

- **Service Composition/Assemblies**: service composition, QoS composition, SLA composition, etc.

- **Service Management**: support for discovering, introspecting, securing, and invoking resources, management functions, measurement, performance indicators, management infrastructure services and toolsets.

- **Service Development Life Cycle (or Service Design and Development)**: service analysis, design methodologies, implementation techniques, construction and testing, provisioning, deployment, execution and monitoring, business process modelling tools.

- **Cross-cutting concerns**: QoS, semantics, non-functional characteristics, security, business transactions, etc.
Agenda

- Service Oriented Computing Research Roadmap
- Theme#1: Service Foundations
- Theme#2: Service Composition/Assemblies
- Theme#3: Service Management
- Theme#4: Service Engineering
- Summary
The requirements to provide an appropriately capable and manageable integration infrastructure for Web services & SOA are coalescing into the concept of the Enterprise Service Bus (ESB).

There are two key ideas behind this approach:

- loosely couple the systems taking part in the integration and
- break up the integration logic into distinct easily manageable pieces.
What is an Enterprise Service Bus?

- ESB is a standards-based IT backbone that leverages Messaging Oriented Middleware functionality throughout the entire business value chain, connecting heterogeneous components and systems.

- A way of building and deploying enterprise SOAs

- Combines features from different types of middleware into one package

- Facilitates the deployment of Web services and solves integration problems

- Key features
  - Web Services: SOAP, WSDL, UDDI
  - Event-based, asynchronous delivery
  - Transformation
  - Routing: Publication/Subscribe, content-based, itinerary

- Platform neutral
  - Connect to anything in the enterprise: Java, .NET, legacy, WS-*
Enterprise service bus connecting diverse applications & technologies

Service orchestration – based custom applications

Portals

Reliable Asynchronous Secure Messaging

Distributed query engine

Adapters

Web Services

JMS/J2EE

MQ gateway

Data sources

Enterprise applications

Multi-platform support

Service interface

service container
The Container Model

- The basic (core) functions of the container are as follows:
  - establishing connectivity and Message Exchange Patterns (MEPs)
  - providing support and provision facilities such as transactions, security, performance metrics, etc, in a declarative and composable manner,
  - providing support for dynamic configuration,
  - monitoring of internal behaviour and state to management systems (services)
  - performing data and protocol adaptation,
  - providing support for services discovery.
Key capabilities of an ESB

- Service Communication Capabilities
  - Ability to route service interactions through a variety of protocols, and to transform from one protocol to another.

- Dynamic Connectivity Capabilities
  - Ability to connect to Web services dynamically without using a separate static API or proxy for each service.

- Topic and Content-based Routing Capabilities
- Endpoint Discovery with Multiple QoS Capabilities
- Leveraging Existing (Legacy) Assets
- Integration & Transformation Capabilities
- Security Capabilities
- Business Orchestration & Long Running Transaction Capabilities
- Management & Monitoring Capabilities
- Scalability Capabilities
ESB Components

Clients

Internet

DNS Router

WSDL/SOAP

Service client

Topic-based and Content-based Publish and Subscribe Mechanisms
Process Orchestration, Transformation, Security, Management, Transport
(WS-ReliableMessaging, WS-Notification, WS-Topics)

ESB Components

interface

Adapter

Proprietary interfaces

Legacy Applications

ERP System

Data Warehouse

Existing EIS & Middleware Platforms
Scaling services in an enterprise service bus.
Research Challenges

- **Requirements for the foundations/container:**
  - is the container model appropriate for SOC?
  - dynamic configuration
  - provisioning mechanisms, e.g., security, transactions, etc.
  - discovery
  - monitoring
  - composition of policies
  - end-to-end QoS

- **Dynamically (re-)configurable run-time architectures:**
  - The run-time service infrastructure should be able to configure itself and be optimized automatically in accordance with specific application requirements & high-level policies (representing business-level objectives)
  - Plug-in Architecture to deal with extensible set of QoS properties:
    - How do we deal with: end-to-end security solutions, multiple SLAs, business transactions, flexible pricing schemes, etc.

- **Support for Evolvable Agreements and Negotiation**
Research Challenges (cntd)

- Topic and content-based routing capabilities: run-time infrastructure should be equipped with routing mechanisms to facilitate not only topic-based routing but also, more sophisticated, content-based routing.

- Infrastructure support for:
  - Application integration
  - Data integration
  - Process integration

- Semantically enhanced service discovery: use of automated means for accurate discovery of services in a manner that demands minimal user involvement.
Humans can publish “themselves” as services, e.g.,:
- Review service
- Consulting service (consultant pattern)

These services are integrated into activities, e.g.,:
- ”send for review“
- ”get expert opinion“

New opportunities for (human) interaction pattern discovery through improved semantics


SOA Model intends 3 actors using *Find-Bind-Execute* cycle:
- Provider *registers* the service
- Consumer *finds* the service...
- ...*binds* to the service provider
- ...and *executes* the service

SOA Practice: Often no registry
- Public registries did not succeed
- Existing registry standards (UDDI and ebXML) are too heavy-weight
- ”Best practice“: Exchange WSDLs and endpoints per email
Our research – VRESCO

- Current registry standards did not succeed
  - Registries should be more than just a lookup service

- Dynamic SOA needs support for:
  - Dynamic (late) Binding: Linking abstract services to concrete service implementations
  - Service Composition using Quality of Service (QoS) attributes and metadata (e.g., pre-/post-conditions)
  - Notifications when certain events occur (e.g., new service is published, QoS of services change, etc.)
  - Service Versioning, Search, Service Metadata, Complex Event Processing, etc.

Agenda

- Service Oriented Computing Research Roadmap
- Theme#1: Service Foundations
- Theme#2: Service Composition/Assemblies
- Theme#3: Service Management
- Theme#4: Service Engineering
- Summary
State of the Art

- On the industry front: Initiatives for developing business process definition specifications, which aim to define and manage business process activities and business interaction protocols comprising collaborating services (“orchestration” and “choreography”).

- On the research front activities have mainly concentrated on:
  - dynamic compositions
  - modularizing compositions
  - enhancing service descriptions (with, for instance, compositional assertions) so that compositions can be assessed and formally verified &
  - on providing context aware services to enable compositions.
  - In the AI field work is mainly in the area of applying AI planning techniques to automate the retrieval and composition of Web services.

- Lack of support for the evolution, adaptation & versioning of business processes.
Research Challenges

- Composability analysis for replaceability, compatibility, and conformance/compliance for dynamic and adaptive processes.

- Adaptive and emergent service compositions.

- Autonomic composition of services:
  - **Self-configuring** compositions e.g., composite services capable of automatically discovering new partners to interact with, automatically select among available suppliers, to choose among different options available for contracts, etc.
  - **Self-optimizing** service compositions that automatically select partners and options to maximize benefits and reduce costs.
  - **Self-healing** compositions to automatically detect that some business composition requirements are no longer satisfied by the implementation and react to requirement violations.
  - **Self-adapting** service compositions are able to function in spite of changes in behaviours of external composite services, they should reduce as much as possible the need of human intervention for adapting services to subsequent evolutions.
Research Challenges (cntd)

- QoS-aware service compositions: to be able to understand & respect each other’s policies, performance levels, security requirements, SLA stipulations, and so forth.

- Separation of business-driven versus systems-level compositions:
  - business-driven automated compositions should exploit business and system level separation in service compositions.
Our Research Methodology

- Emerging problems
- Runtime behavior
- Metrics/model dependencies
- Performance and reliability

- QoS model
- Interaction patterns and metrics
- Context model and sharing
- Service evolution model
- Human/team metrics
- Trade-off models
- Domain-specific languages

- Service Composition
- Service selection
- Activity ranking
- Human selection
- Reliable processes
- Process mashup

- Individual service
- Flows and processes
- Teamwork
- Middleware
- In-situ adaptation
# Service Evolution

<table>
<thead>
<tr>
<th>Category</th>
<th>Biological Evolution</th>
<th>Web Services Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic element</td>
<td>Individual</td>
<td>Web Service</td>
</tr>
<tr>
<td>Representation of information</td>
<td>Genomic variant</td>
<td>Functional and non-functional properties (e.g., QoS, SLA, licensing, service interface), requirements, feedbacks, etc.</td>
</tr>
<tr>
<td>Factor of influence</td>
<td>Food, water, etc.</td>
<td>Hosting environment, integrator, user, provider, developer</td>
</tr>
<tr>
<td>Evolutionary mechanism</td>
<td>Mutation, natural selection, recombination</td>
<td>Service replication, derivation and composition, service selection and utilization, mashup</td>
</tr>
</tbody>
</table>

---

Agenda

- Service Oriented Computing Research Roadmap
- Theme#1: Service Foundations
- Theme#2: Service Composition/Assemblies
- Theme#3: Service Management
- Theme#4: Service Engineering
- Summary
Service Mgt & Monitoring

- Composite service developments need mechanisms that provide insights into the health of systems that implement Web services & into the status and behaviour patterns of loosely coupled applications.
  - Failure or change of a single application component can bring down many interdependent enterprise applications.
  - The addition of new applications or components can overload existing components, causing unexpected degradation or failure of seemingly unrelated systems.

- Application performance depends on the combined performance of cooperating services & their interactions.
To counter such situations, enterprises need to constantly monitor the health of their applications.

- The performance should be in tune at all times and under all load conditions.

Service management spans a range of activities from installation and configuration to collecting metrics and tuning to ensure responsive service execution:

- Service-Level Agreement (SLA) negotiation, management, auditing, monitoring, and troubleshooting, service lifecycle/state management, performance management, services and resources provisioning, & aspects like scalability, availability and extensibility and others.
State of the Art

- Service operations management gathers info about:
  - the managed service platform, services and business processes and managed resource status and performance, and supporting specific management tasks (e.g., root cause failure analysis, SLA monitoring and reporting, service deployment, and life cycle management and capacity planning).

- Service operations management
  - provides global visibility of running processes, comparable to that provided by Business Process Management tools.
  - defines & supports active capabilities versus traditional passive capabilities e.g., rather than merely raising an alert when a given service is unable to meet the performance requirements of a given service-level agreement, is able to take corrective action itself.
  - provides global visibility of running processes, comparable to that provided by Business Process Management tools.
WS Distributed Mgt is an interoperability protocol mgt info & capabilities in a distributed environment via Web services. WSDM focuses:

- Management Using WS (MUWS) uses Web services technologies as the foundation of a modern distributed systems mgt. It defines how to describe manageability capabilities of resources using WSDL documents.
- Management of WS (MOWS) addresses the specific requirements for managing Web services themselves just like any other resource.
State of the Art (cntd)

- On the research front activities have mainly concentrated on:
  - on assessing the impact of service execution from a business perspective and, conversely, to adjust and optimize service executions based on stated business objectives.

- Service mgt entails monitoring. Here research activities focus on:
  - dynamic monitoring techniques that are capable of employing monitoring rules governing the control of composite services, e.g., BPEL processes.
  - capturing and monitoring negotiations that incorporate security policies and policy models that facilitate service life-cycle management.

- Also research on QoS metrics for selecting Web services and for establishing trust between trading partners.
Research Challenges

- Autonomic management of services:
  - Self-configuring mgt services configure themselves automatically to adapt to different environments & optimize for particular kinds of use.
  - Self-adapting mgt services adapt dynamically to changes in the environment, market and so on, using policies provided by the distributed platform administrator.
  - Self-healing mgt services can discover, diagnose and react to disruptions. Corrective action could involve a product altering its own state or effecting changes in other components in the environment.
  - Self-optimizing mgt services can monitor and tune resources automatically to meet end-user or business needs, e.g., reallocating resources in response to dynamically changing workloads to improve overall utilization, or ensure that business transactions are completed in a timely fashion.
  - Self-protecting management services can anticipate, detect, identify and protect against threats, e.g., unauthorized access and use, virus infection and proliferation, etc & take corrective actions to make themselves less vulnerable.
Autonomic Services – our approach
Autonomic Service Adaptation

Services react (passive) to changes and to anticipate (pro-active) changes

- Based on BPEL monitoring and runtime services adaptation (e.g., degradation of QoS, based on replacement strategies and transformers)


- Based on activity patterns (mining of activities)


- Based on service patterns (e.g., mining of service dependencies)

Finding Patterns in Ad-hoc Team Interactions

- **Proxy**
  - 1:1 relation to original
  - e.g., secretary, assistant

- **Broker**
  - e.g., person who is responsible to answer all client requests

- **“Master/Slave”**
  - sending identical requests to multiple recipients
  - e.g., multiple participants are requested to state their cost estimates

- More patterns…(ICEIS 2007)

Service Interaction Mining

- Different scopes/levels for mining
  - (1) Individual Services
  - (2) Service Selection (A or B)
  - (3) Service Dependencies (a set of services is always used in combination with each other)
  - (4) Workflow Mining (activities in a process)

Context is a beast

Context Tunneling – Context Scopes

- Based on our work in the inContext project http://www.in-context.eu/

- Tunneling is based on three notions of context scopes:
  - Individual Context
  - (Complex) Activity Context
  - Team Context

- We need “integration” of context (e.g., individuals collaborate on shared activities) -> processes
Context Management: distributed storage

- Context information collected from different sources
- Centralized context store is not suitable
- Context information is stored in different services
  - Linked through a core model
WORKPAD EU Project

EU STREP FP/ WORKPAD
Pervasive Software Environments for Supporting Disaster Responses

WORKPAD Scenario
(IEEE Internet Computing 1/2008)

Vimoware: Vienna mobile middleware

Context and Interaction

Mining interactions in disaster scenario
Agenda

- Service Oriented Computing Research Roadmap
- Theme#1: Service Foundations
- Theme#2: Service Composition/Assemblies
- Theme#3: Service Management
- Theme#4: Service Engineering
- Summary
Each process has deep knowledge and value for an enterprise
- Need to study and focus but there are generic properties also

Business processes
- Measures of success
- Control features
- Formal properties and descriptions
- Real-world constraints
- Decomposition and composition
- Contractual and expectations of cost, quality, reliability
- Can be internal or external to an organization

Business process examples
- Procurement
- Order & Sales management
- HR pension management
- Electronic chip design
Developers in their early use of SOA implement a thin SOAP/WSDL/UDDI veneer on top of existing applications or components & leave the underlying component untouched.

Conventional s/w development methodologies such as Object-Oriented Development & Component Based Development do not address the three key elements of an SOA:

- services,
- service assemblies (composition), and
- components realizing services.

Early research activities concentrate on how to provide sufficient principles & guidelines to specify, construct and refine and customize business processes from a set of internal and external services.
Research Challenges

- Engineering of Service Compositions:
  - Associating a services engineering methodology with standard s/w development & business process modelling techniques:
    - NOT UML!!
    - Rational Unified Process - analysis and design of iterative software development.
    - Supply Chain Operations Reference (SCOR) Framework (standard guidelines for companies that examine the configuration of their supply chains, identify & measure metrics in the supply chain).
  - Automated, transparent user centred support to the entire business process lifecycle of composed services.
  - Design techniques for managing service versioning and adaptivity.
  - Service Governance to govern end-to-end business processes that are composed out of variety of service fragments that need to be maintained separately by different organizations.
Business Process Decomposition

- Focus on defining business services from the business process point of view.

- Start with some business process & decompose it into increasingly smaller sub-processes. The resulting smallest sub-processes then become candidate business services for implementation.
  - The more business processes decomposed in this way, the more commonality across their sub-processes is achieved, thus building an appropriate set of reusable Services.

- Simultaneously take existing business logic ingrained in app. code & expose it as services that specify not the overall business process, but rather the mechanism for implementing it. This yields two categories of Services:
  - business functionality services that are reusable across multiple processes, and fine-grained
  - utility services that can provide value to various services across the organization.
Service granularity refers to the **scope** of functionality exposed by a service. Services may come at two levels of granularity:

- A **coarse-grained** interface might be the complete processing for a given service, e.g., “SubmitPO” - the message contains all business information needed to define a purchase order.
- A **fine-grained** interface might have separate operations for: “CreateNewPO”, “SetShippingAddress”, “AddItem”, etc.

**Fine-grained services** might be services that provide basic data access or rudimentary operations. These services are of little value to business applications.

**Coarse-grained services** are composed from finer grained services.
Service Granularity Concerns II

- The frequency of message exchange is an important factor. Sending & receiving more info. in a single request is more efficient than sending many fine-grained messages.
  
  - Several redundant, fine-grained services leads to increased message traffic & tremendous overhead & inefficiency.
  - A small collection of coarser-grained services - each of which implements a complete business process - that are usable in multiple scenarios is a better option.

- Heuristics identify the right level of granularity for services, e.g., clearly identifiable business concepts, highly usable and reusable concepts, concepts that have a high-degree of cohesion and low-degree of coupling, etc.

- Vertical sectors, e.g., automotive, travel industry, etc, standardize business entities & processes by choosing their own levels of granularity.
Agenda

- Service Oriented Computing Research Roadmap
- Theme#1: Service Foundations
- Theme#2: Service Composition/Assemblies
- Theme#3: Service Management
- Theme#4: Service Engineering
- Summary
COMPAS addresses a major shortcoming in today’s approach to design SOAs:
- Throughout the architecture various compliance concerns must be considered

Examples:
- Service composition policies, Service deployment policies,
- Information sharing/exchange policies, Security policies, QoS policies,
- Business policies, Jurisdictional policies, Preference rules
- Intellectual property and licenses

So far, the SOA approach does not provide any clear technological strategy or concept of how to realize, enforce, or validate them
Compliance Concerns: Examples

- Service composition policies
  - e.g., use of data for a certain case only
- Service deployment policies
  - e.g., geographic restrictions for service instances
- Information sharing/exchange policies
  - e.g., request/response use specific message type
- Security policies
- QoS policies
- Business policies
- Jurisdictional policies
- Preference rules
- Intellectual property and licenses
Compliance Concerns Categories

- **Technical compliance concerns that must be validated at design time**
  - e.g., compliance to composition policies

- **Technical compliance concerns that must be validated at runtime**
  - e.g., compliance to QoS policies

- **Domain-oriented compliance concerns that must be validated at design time and runtime**
  - This category is the most complex one, as it is build on top of the two technical categories
  - e.g., compliance to preference rules, to licenses, etc.
Problem in Detail

- A number of approaches, such as business rules or composition concepts for services, have been proposed.
- None of these approaches offers a unified approach with which all kinds of compliance rules can be tackled.
- Compliance rules are often pervasive throughout the SOA.
- They must be considered in all components of the SOA.
- They must be considered at different development times, including analysis time, design time, and runtime.
Current Practice for Dealing with Compliance

- In many cases, business compliance today is reached on a per-case basis
- Companies do not have a generic strategy for business compliance
- Instead they use ad hoc, hand-crafted solutions for specific rules to which they must comply
Issues with the Current Practice

- Systems are
  - hard to maintain
  - hard to evolve or change
  - hard to reuse
  - hard to understand
- It is difficult to ensure **guaranteed compliance** to a given set of rules and regulations
- It is difficult to keep up with **constant changes** in regulations and laws
Process-driven development scenarios

Business/domain experts

BPMN diagram

UML Activity diagram

EPC diagram

IT experts

WSDL + BPEL

Java/J2EE
Developer’s nightmare – numerous tangled process concerns
Motivation: interoperability and reusability of processes

- Why process description (re)use is difficult
  - The integration of many tangled aspects hinders understandability, modularity
    - the control flow, service interactions, message and message types, fault handling, transactions, compliances, process engine configurations, etc.
  - Stakeholders have different point of views, abstraction levels, skill sets, needs, etc.
    - Business or domain experts
    - IT experts: developers, administrators
Existing solutions so far?

- Manually done by "copy and paste"
  - Time-consuming and error-prone
  - Only IT developers are able to do "copy and paste"
  - Hinders scalability and agility

- Language transformation-based approaches
  - E.g., BPMN/EPC/UML Activity diagrams to BPEL/WSDL, and vice versa.

- Workflow mining
  - Extracts (only) workflows from applications
Proposed solution: View-based Model-driven Engineering

- Separation of concerns principle
  - A realization: concept of architectural views
- Model-driven engineering
  - (semi)-formalization of process fragments to enhance modularity, reusability, etc.
  - separation of abstraction levels by tailored views to enhance adaptability, understandability
- View-based reverse engineering
  - extracts (semi)-formalized views
Proposed integration solution

High-level (domain-specific) concepts, designs

Process (technology-specific, platform-specific) implementations

Modeling Framework
**View-based integration architecture**

- **View-based Modeling Framework**

- **View-based Interpreters**
  - defines Meta-models
  - produces High-level Views
  - produces Low-level Views

- **Meta-models**
  - conforms to High-level Views
  - conforms to Low-level Views

- **High-level Views**
  - refines Meta-models
  - corresponds to High-level Languages

- **Low-level Views**
  - refines Meta-models
  - corresponds to Low-level Languages

- **High-level Languages**
  - described in Meta-models

- **Low-level Languages**
  - described in Meta-models

- **Process descriptions**
  - interpreted by View-based Interpreters
  - described in Meta-models

- "virtual" integration of level and high low-level representations in various languages
View-based Modeling Framework (VbMF)
Separation of concerns
Separation of technical and domain-oriented views
View-based Modeling Framework (VbMF)

Horizontal extensibility of VbMF

Information View meta-model

Collaboration View meta-model

Control-flow View meta-model

Transaction View meta-model
View-based Modeling Framework (VbMF)

Information View meta-model

An extension of the Information View meta-model

Vertical extensibility of VbMF

High-level abstraction

Lower-level abstraction
View-based Interpreters

Chain of Responsibility and Partial Interpreter Pattern - General Approach For View Extraction
The overall Tool chain

- **Design**
  - Modeling of Views within the VbMF (EMF/GMF-Editor)

- **Transformation**
  - Invocation of the Code Generation (oAW-Workflow)

- **Validation**
  - Semantic validation/optimization of a process

- **Deployment**
  - Prepare and deploy process on a BPEL engine

- **Execution**
  - Fire & Forget a long-running process
How views are reused?

- Business, domain experts
- IT experts

High-level, problem-oriented views
- View-based repository

Low-level, technology specific views
- Code generation
- Executable code, configurations
Current Work

- Collaborative Model-Driven Development
  - Lightweight Collaborative Model-Driven Environment
  - Correlation of Process Stakeholders & MDD Artefacts
  - Model Repository

- Distributed Process Monitor
  - for Debugging, Logging, Monitoring, etc.
  - Publishers = Components
  - Broker Architecture
  - Distributed Subscribers
Current Work

- Dependency management
  - Traceability
  - Change impact analysis
  - Change propagation
- Collaborative view repository
  - View-based, model-driven repository
Summary of our approach

Separation of Concerns (e.g., architectural views) to master the complexity

Process-driven SOA

SOA

Model-Driven Engineering

VbMF

provides an efficient method for integrating business functionality
reconciles the heterogeneous nature of software systems
Summary of research roadmap

- Research activities in SOC are very fragmented. This necessitates that a broader vision and perspective be established—one that permeates and transforms the fundamental requirements of complex applications that require the use of the SOC paradigm.

- The SOC research roadmap launches four pivotal, inherently related, research themes to SOC:
  - service foundations,
  - service composition,
  - service management and monitoring, and
  - service-oriented engineering.
Conclusion

- Autonomic and adaptive Services and environments -> shift to runtime considerations

- Novel abstractions needed to model, monitor, predict, and execute compositions and mashups (Top-down and Bottom-up approaches)

- Interaction Mining and Pattern detection of activities and services

- Context (reuse across services/applications)
Thanks for your attention!

Prof. Dr. Schahram Dustdar
Distributed Systems Group
Institute of Information Systems
Vienna University of Technology
Austria

http://www.infosys.tuwien.ac.at/Staff/sd/