BPM & SOA
SOA

- **Service-Oriented Architecture (SOA):**
  - Paradigm for structuring information & sw systems based on capabilities that part of a system offers to other parts
  - Services have higher-level of abstraction & underlying philosophy than components (delivery)

- **OASIS Definition:**
  - SOA is a paradigm for organising & utilizing distributed capabilities that may be under the control of different ownership domains
  - Capability wrt. business as well as specific application systems
  - Service-orientation relevant at both the business & technical level
  - SOA provides common abstractions & principles to structure systems uniformly from the IT & business perspective
SOA

- Service is a business concept but turned also as IT concept:
  - Close gap between business and IT & achieve higher degree of business-IT alignment

- Ownership:
  - Services must be delivered to exist
    - Resource encapsulated by a service should exist at particular location & must be maintained and managed by service provider so as to deliver a capability to a service consumer
  - Service providers & consumers operate independently & can exist in different ownership domains
    - Perfect fit for SOA and BPM
    - BP can span different functional domains
    - SOA aims at structuring system such that eases communication & handover between these domains
WHAT ARE WEB SERVICES?

- A web service is any **piece of software** that makes itself **available over the internet** and uses a standardized **XML** messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.

- Web services are **self-contained, modular, distributed, dynamic applications** that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as **TCP/IP, HTTP, Java, HTML, and XML**.
WHAT ARE WEB SERVICES?

- The basic web services platform is XML + HTTP. All the standard web services work using the following components:
  - **SOAP** (Simple Object Access Protocol)
  - **UDDI** (Universal Description, Discovery and Integration)
  - **WSDL** (Web Services Description Language)
  - **REST** (REpresentational State Transfer)

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, SOAP and REST. A web service takes the help of:

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.
- REST is an *architectural style* and a *design* for network-based software architectures.
Enabler for BPM

- Services provide an abstraction to bridge BP activities with underlying resources & capabilities
- Entire BPs can be exposed as services to be consumed by customers or be plugged into other BPs
- Service can serve as an entry point for BPs

To design & link services with BPs

- Service granularity must be considered:
  - Service at level of individual BP activities or
  - at level of long-running BPs
SOA ARCHITECTURE PRINCIPLES

- W3C Definition:
  - Service is an abstract resource that represents a capability

- Capability is offered by a service provider by performing a set of actions on behalf of service consumer at some time & place and interacting with consumer via a particular channel

- Service bus is a medium connecting service provider & consumer and comprises various technical infrastructure elements

- Service repository facilitates discovery of services & provides additional service information (constraints & service levels)
OASIS indicates that specific SOA aspects must be considered when analyzing and designing services for interaction, including visibility & interaction.

- Service provider & consumer must interact independently of whether they are humans or automated programs.
- Service consumer needs to know the service I/O & actions that can be performed against the service, as part of service description, for successful interaction.

Five principles apply for the identification of services:

- Contract orientation
- Cohesiveness
- Coupling
- Reusability
- Autonomy
Contract orientation:

- Service must share a formal contract (Service Level Agreement –SLA) defining terms of information exchange & commitments made by both parties (provider & consumer) to define a (business) relationship.

- Contract encompasses the description of:
  - Functional & non-functional characteristics
    - Includes description of exposed operations to be invoked
  - Remedies when violation of commitments occur
  - Ways to monitor the commitments by which party

- Trust between parties is increased
  - Participation in contract of third parties for handling subtle points
SOA ARCHITECTURE PRINCIPLES

- **Cohesiveness:**
  - Refers to the concept of grouping operations when they are functionally related to the performance of a task.
  - Analysis of underlying business objects is indicator of cohesiveness.
    - High relation of operations on a business object indicates high cohesiveness.
    - If operations of two services are highly-related, services can be merged.

- **Reusability:**
  - Service should be useful in different cases / circumstances / scenarios & be exploited by different consumers.
SOA ARCHITECTURE PRINCIPLES

Coupling:

- Describes strength of interdependency between multiple services & service compositions
- Independent services are more reusable & maintainable
- Coupling between services must be as loose as possible
  - Otherwise services must be merged
- Levels of dependency can be minimized by minimizing the interactions between services
- Balance between cohesion & coupling must be discovered
- Coarse-grained interaction preferable than fine-grained one
  - Bigger size of data exchanged leads to less interactions
**SOA ARCHITECTURE PRINCIPLES**

- **Autonomy:**
  - Level of independence of a service
  - A purely autonomous service has **full control** of its **environment**
    - Increased reliability & predictability as external unpredictable influences are minimized
  - **Data normalization** techniques can be exploited to design operations in a **non-redundant** manner
Business service:

- is an outcome of set of operations of an organisation
- can represent operations at different levels
- can be aligned along the organisation hierarchical structure or be based on actual business capabilities & domains
- may or may not leverage existing IT infrastructure
  - Distinguishable from a software service
Software service:

- is part of an application system that can be consumed separately by different entities
- may enable a business service or can provide a capability that contributes to a business service delivery
- It can also have a non-business but technical scope
- Can be distinguished into:
  - Business-related services identified & specified based on business requirements
    - Requirements may refer to BPs, tasks or business entities
  - Technical services that are business-logic agnostic & include utility services that offer generic functions to other software services
<table>
<thead>
<tr>
<th>TYPES OF SERVICES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service-Type</strong></td>
</tr>
<tr>
<td><strong>Granularity</strong></td>
</tr>
<tr>
<td><strong>Composition</strong></td>
</tr>
<tr>
<td><strong>Interaction</strong></td>
</tr>
<tr>
<td><strong>Exchange Patterns</strong></td>
</tr>
<tr>
<td><strong>State</strong></td>
</tr>
<tr>
<td><strong>Accessibility</strong></td>
</tr>
</tbody>
</table>
## TYPES OF SERVICES

<table>
<thead>
<tr>
<th>#</th>
<th>SOAP</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A XML-based message protocol</td>
<td>An architectural style protocol</td>
</tr>
<tr>
<td>2</td>
<td>Uses WSDL for communication between consumer and provider</td>
<td>Uses XML or JSON to send and receive data</td>
</tr>
<tr>
<td>3</td>
<td>Invokes services by calling RPC method</td>
<td>Simply calls services via URL path</td>
</tr>
<tr>
<td>4</td>
<td>Does not return human readable result</td>
<td>Result is readable which is just plain XML or JSON</td>
</tr>
<tr>
<td>5</td>
<td>Transfer is over HTTP. Also uses other protocols such as SMTP, FTP, etc.</td>
<td>Transfer is over HTTP only</td>
</tr>
<tr>
<td>6</td>
<td>JavaScript can call SOAP, but it is difficult to implement</td>
<td>Easy to call from JavaScript</td>
</tr>
<tr>
<td>7</td>
<td>Performance is not great compared to REST</td>
<td>Performance is much better compared to SOAP - less CPU intensive, leaner code etc.</td>
</tr>
</tbody>
</table>
Service can be elementary or composite

- Elementary services can be classified into task (logic-driven), entity (data-driven) & utility services
- Composite services can be classified into data-aggregation services & process-driven composite services

Services can be differentiated according to interaction style, information exchange patterns, state information management & intended customer types (external, internal or both)

Utility services:

- Business-logic agnostic
  - Provide re-usable cross-cutting functionalities related to processing data within legacy application environments
**TYPES OF SERVICES**

**Entity services:**
- Creation & management of business entities
- CRUD operations over business entities complying with business rules
- Enforce a vertical layering of data
  - One major business entity is usually dealt with

**Task services:**
- Directly related to business tasks
- Modelled for specific processes
- Contain specific business logic
- Encompass business rules & functionality provided centrally in a consistent manner across the organisation
Composite services:

- Parent controller of a number of entity, task & utility services
- Encapsulate process logic
- Control & maintain the state of the process
- Process-driven composite service rely on service orchestration
Service-enabled processes can be modelled either via service orchestrations or choreographies.

Service choreography:
- **Global model of interactions** that can occur between a set of services in the context of a service-enabled process.
- Not only interactions but also **dependencies** between them are captured, including control-flow, data-flow, timing & QoS dependencies.
- **High-level view** of service-enabled process:
  - Does not capture internal actions involved in a service whose effect is not externally visible.
  - **Global perspective** is provided: view of a observer and not a process participant.
  - Services are **abstract**, may not correspond to an actual service deployed on particular end-point.
CHOREOGRAPHY EXAMPLE
For each participant/role, a contract is expected to be fulfilled wrt the service(s) offered:

- Includes description of messages to send/receive & relations between these messages
- Messages carry information about business entities
- Behavioural interface is called the view of a choreography based on a particular role:
  - Service operations & their I/O (message types) plus way multiple services are related in the context of a BP
  - Structure of interactions & ordering dependencies between these interactions
SERVICE ORCHESTRATION

- Refinement of a behavioural interface
- Includes the interactions plus internal actions that a service must perform
- Lower-level and focused-view of a service-enabled process
- Can be further refined into an executable BP e.g. in BPEL
SERVICE ORCHESTRATION EXAMPLE
Two approaches based on different model type chosen

Choreography-driven service design approach:
1. Design a choreography capturing all interactions between roles in the context of an end-to-end collaborative BP
2. Derive the behavioural interface of the involved services
3. Refine behavioural interfaces into service orchestrations

Orchestration-driven service design approach:
1. Define an orchestration of service-enabled BP to fulfil a certain goal
2. Find appropriate sub-services to fill-in the orchestration
3. Derive an interface from the orchestration (view of orchestration without internal actions) -> service & interface exposed for further composition into a broader system
SERVICE IDENTIFICATION METHODS

**Domain-driven methods:**
- Utilize business models, enterprise architecture models or domain models to identify capabilities to be exposed as services.
- Identify what is the business of an organisation & then define accordingly service boundaries.
- The high-level services identified can be decomposed until the level of elementary software services.

**Process-driven methods:**
- Business process models are mapped to business services.
- Based on the model information (flow of info & objects) software services can be identified to be realized by IT.
SERVICE IDENTIFICATION METHODS

○ Entity-driven methods:
  - Rely on models specifying business entities
  - Entity models, class diagrams, information models or taxonomies can assist in identifying services operating over business entities

○ Reference models:
  - High-level models assist in identifying service boundaries
  - Such models are typically applicable in multiple scenarios & contexts so do not reflect specific organisational requirements
  - Must be coupled with & mapped to organisation-specific characteristics for proper service identification
Hybrid methods:

- Combine two or more previous methods
  - E.g., domain-driven & entity-driven
- Goal-modelling can be integrated to identify & eliminate service redundancy
Relationship between processes & services:

- Process is a logical sequence of activities towards a business goal
- Activities are invoked in terms of message exchanges
- Activities are logically-grouped into services (elementary or composite)
- Execution of composite service is driven by a process which can require certain operations to be delivered by other services
PROCESS-DRIVEN SERVICE IDENTIFICATION
Presented method is consolidation of existing methods

- Differences are identified & consolidated based on SOA principles
- Assumption: Scope of service identification is already defined:
  - BPs and areas of organisation to be enabled via services
- Method comprises 7 steps
I. Analyze visibility & handover of process steps:

- Most elementary process steps are discovered/selected

- Process is then analyzed according to its visibility & interaction potential:
  - Line of interaction: specifies parts or functions of BP to be undertaken by customer
  - Line of visibility: process visibility degree specification – stakeholders may comprise external business partners and internal actors

- Potential grouping of functionality is the outcome to be exposed as a service
2. Identify Entity Services:
   - Previous method step outcome as input
   - **Boundary** of entity service identified through *analysis of service context*
     - Entities are discovered in the *examined process model* as well as the *operations* operating on them

3. Identify Potential Service Operations:
   - Each *process step* regarded as potential *service operation*
     - Manual tasks or process *steps* mapping to *legacy systems* are excluded
4. Define Logical Context(s):

- Remaining process steps grouped according to logical context (thus actually defining service boundary)

- Service cohesion plays important role
  - Group operations functionally related

- Service reusability is employed to specify further operations within the boundary of a service
  - Operations that have high potential to be re-used in different scenarios
  - Entity services can be potentially adjusted (CRUD ops must be created)

- Coupling is used to identify sequential dependencies between operations
4. Define logical context (cont.):

- Service components can be identified for recurring sequences of service operations or depending on new logical contexts
- Operations that can be executed in different timelines by customers can also map to separate services
- Finally, purely technical and logic-agnostic services are identified

5. Define compositions:

- Scenarios are used to check for service composition & consolidation wrt the identified task, entity & utility services
- The appropriateness of service boundaries is evaluated & discovery of missing logic to be shifted in task or composite services
- Composite services can be exposed to a specific stakeholder set
6. Detail the operations:

- Detail service operations to identify potential for enhancements
  - Service operation I/O is specified
  - Aim is to maximize cohesiveness & minimize operation coupling
  - Reusability in terms of mapping operations with concrete input to operations with a generic input relative to business requirements is considered
7. Perform mapping:
   - For each operation candidate identify processing requirements, especially the application logic
   - Identify if application logic already exists or can be delivered by a third-party
   - Application logic requirements could be broken-down into steps mapping to more fine-grained operations to be clustered according to the principles of cohesion & autonomy (e.g., when mapping to the same legacy system)
   - If new services or operations are identified, service compositions can be revisited and might be enhanced
Method links task & composite services to entity services

Different service designers may end up with different services in the end

- Different ontologies or different priorities on SOA principles might be employed
Two approaches have been proposed:
- One based on process maps
- One based on explicit semantic goal modelling

Both approaches also propose ways to map business to software services

Main idea:
- Business service should satisfy an intention or (fine-grained) goal
- Set of business services can lead to the satisfaction on the organisation’s root/high-level goals
GOAL-DRIVEN SERVICE IDENTIFICATION – 1ST APPROACH

- Relies on Intentional Service Model & Process Maps

- Intentional Service Model:
  - Models the different types of intentional services
  - Models 4 main aspects for business/intentional services:
    - Service interface
    - Service behaviour
    - Service composition
    - QoS
GOAL-DRIVEN SERVICE IDENTIFICATION – 1ST APPROACH
Service interface is described through:

- The intention satisfied
  - e.g., make room booking
- The initial & final situation mapping to the I/O parameters structured as business object classes
  - e.g., booking & customer class is input while payment is output

Service behaviour is specified through:

- The pre- & post-conditions constituting the initial & final set of states characterizing the respective initial & final situations
  - e.g., booking.state="ok" && customer.status="registered" -> preconditions, booking.state="paid" && payment.status="done" -> post conditions
GOAL-DRIVEN SERVICE IDENTIFICATION – 1ST APPROACH

Service composition:

- Services can be atomic or aggregate
- Atomic services have intentions satisfied by SOA-level software services
- Aggregate services map to high-level intentions
  - Need to be decomposed into less complex services until the level of atomic (intentional services)
  - Thus, there should be a service composition process which bridges the gap between the actual functionality (at the atomic service level) and high-level perceptions of business executives for services fulfilling their (strategic/tactical) intentions
Service Decomposition follows intentional model

- The component services should map to goals whose combination satisfies the goal/intention of the composite service

- In an AND goal decomposition, 2 operators are used to combine component services, \( \bullet \) (sequence) and \( || \) (parallel). * denotes repetitive service execution

- e.g., \( S_{\text{make_confirmed_booking}} = \bullet (S_{\text{make_room_booking}}, S_{\text{accept_payment}}) \)

- Variability in goal modelling leads to flexibility in goal achievement & adaptability in service execution. 3 types of variants are introduced:
  - Alternative (\( \otimes \)) maps to XOR goal relationship:
    \( S_{\text{accept_payment}} = \otimes (S_{\text{accept_payment_electronic_transfer}}, S_{\text{accept_payment_credit_card}}, S_{\text{accept_payment_cash}}) \)
  - Choice (\( v \)) maps to OR goal relationship:
    \( S_{\text{investigate_candidate_booking}} = v (S_{\text{investigate_candidate_booking_on_internet}}, S_{\text{investigate_candidate_booking_travel_agent}}) \)
  - Multipath (\( \cup \)) maps to different compositions that can realize a certain service:
    \( S_{\text{make_confirmed_booking}} = \cup (\bullet (S_{\text{make_room_booking}}, S_{\text{accept_payment}}), S_{\text{get_rewarded_booking}}) \)
GOAL-DRIVEN SERVICE IDENTIFICATION – 1ST APPROACH

- **QoS:**
  - Quality aspects are captured by quality/soft goals
  - Quality frameworks can be used for their specification
  - A quality goal is satisfied through the partial contribution of a service towards this goal
  - Simple QoS -> quality of atomic service
  - Global QoS -> quality of aggregated service
GOAL-DRIVEN SERVICE IDENTIFICATION – 1st APPROACH

- **Simple QoS:**
  - **Dually expressed** with a qualitative & quantitative part
  - **Qualitative evaluation** maps to *degree of goal satisfaction*. The following values are allowed: “Very Satisfied” -> “++”, “Satisfied” -> “+”, “Not Satisfied” -> “-”, “Not at all Satisfied” -> “--”
    - e.g., QoS(S_{accept\_payment}) = (<Performance,++>, <Confidentiality,++>)
  - **Quantitative evaluation** relies on assigning indicators to each soft goal indicating the *quantitative satisfaction degree of the goal*. Each indicator has reference values which map to the qualitative values.
    - e.g., confidentiality is mapped to fraud\_rate & security\_level indicators. If fraud\_rate < 0.02, then satisfaction degree is “++”. If fraud\_rate is >= 1, then satisfaction degree is “-”.

CS 565 - LECTURE 8

3/4/2019
GOAL-DRIVEN SERVICE IDENTIFICATION – 1\textsuperscript{ST} APPROACH

- Global QoS:
  - Calculate global QoS based on QoS of component services
  - In case the $\textbullet$ operator is involved, then:
    - $q \text{ (service)} = \min(<q, cs_1>, <q, cs_2>, \ldots, <q, cs_n>)$
  - In case the $\otimes$ operator is involved, then:
    - $q \text{ (service)} = \max(<q, cs_1>, <q, cs_2>, \ldots, <q, cs_n>)$
  - In case of a composition where different qualities with different satisfaction degrees are involved per composite service, then:
    - $QoS \text{ (service)} = \text{union}(<q_1, cs_1>, <q_2, cs_2>, \ldots, <q_n, cs_n>)$
Intentional services are stored in a registry

Registry used for service discovery

Discovery relies on first matching intentions & then on conditions and situations matching (i.e., matching both the service interface & behaviour)

MAP formalism is used to represent a business in intentional terms

Then, based on this formalism, guidelines are provided for identifying intentional services
MAP formalism

- Labelled directed graph with intentions as nodes & strategies as edges.
- An edge reaches a node if the respective strategy achieves the node intention. A strategy $S_{ij}$ between two intentions indicates the way the second intention $I_j$ can be achieved if the first $I_i$ is satisfied. The triple $< I_i, I_j, S_{ij} >$ is called section.
- An intention is achieved via executing a process. Two special intentions indicate the start & stop of a process.
- Map as a whole maps to a global intention that indicates the global purpose of the business.
- Sections are connected via 4 relationships: multithread, bundle, path & multipath.
GOAL-DRIVEN SERVICE IDENTIFICATION – 1ST APPROACH

- Multi-thread Relationship:
  - Set of sections indicating different ways a target intention can be achieved from a source one
  - Sections in set are in thread relationship

- Bundle Relationship:
  - Same as previous but exclusive OR instead of simple OR is captured

- Path Relationship:
  - Target intention in preceding section is source in the subsequent one

- MultiPath:
  - Different combinations of sections to achieve a specific intention forming multiple paths in a MAP

- A section of a map can be refined into another map at lower level of abstraction
GOAL-DRIVEN SERVICE IDENTIFICATION – 1ST APPROACH

refinement
Intentional Service Derivation/Composition:

For a specific map $m$, its intention $I_m$ and its associated service $S_{I_m}$:

1. Associate every non-refined section to atomic service
2. Calculate all paths of a map
   - Automatic generation based on adaptation of MacNaugton and Yamada algorithm to operate over maps
3. Determine aggregate services through correspondence between section relationships and service aggregation operators
   - path-$\rightarrow$composite, bundle-$\rightarrow$alternative, multi-thread-$\rightarrow$choice, multipath-$\rightarrow$multipath
4. For each refined section $r$, repeat 1-3 for respective map $m_r$
Atomic intentional services must be operationalized into software services

- By having SW realizing functionality of atomic services, also composite services can be operationalized through SW service compositions

Manual activity in the approach carried out by a SW developer

Guidelines were proposed to go from intentional to logical level and from that to the implementation level to support a two-phase process:

- Atomic intentional service operationalization
- Implementation of operationalizations into SW services
GOAL-DRIVEN SERVICE IDENTIFICATION – 1\textsuperscript{ST} APPROACH
GOAL-DRIVEN SERVICE IDENTIFICATION – 1ST APPROACH

- Atomic intentional service descriptions transformed into operational ones in a platform independent model, namely OSM

- Operational description comprises 3 parts:
  - Interaction: sequence of user interactions
  - Function: business logic
  - Coordination: coordination between the two former by ensuring synchronization of invoked functions and data interoperability

- Transformation performed in two steps:
  - Intention operationalization knowledge capturing through scenarios according to L’Ecritoire approach, where rules are used to map scenario content to a particular structure and reason on it to derive sub-goals such that functionality & variability is explored
  - Scenario description transformation into OSM models according to correspondence between scenario & OSM meta-models
GOAL-DRIVEN SERVICE IDENTIFICATION – 1ST APPROACH

- **Transformation of OSM models to remote portlet implementation according to the** Web Service for Remote Portlets (WSRP) **specification** (presentation-oriented web services):
  - **Interaction part** translated to portlet descriptor (UI service)
    - OSM back office interactions translated to requests on composite web service
    - User interactions translated to JSP views containing the identified back office requests
  - **Coordination part** to a composite web service specification (BPEL + WSDL)
  - **Function part** to atomic web services in WSDL
GOAL-DRIVEN SERVICE IDENTIFICATION – 2ND APPROACH

● Novel business service design method
  ■ Takes view that BS is a hybrid concept:
    ● Intentional: directly linked to business goals
    ● Operational: different software service (SS) compositions can realize it
  ■ Two phases involved:
    1. BS composition to map to more concrete/operationalized/atomic services
    2. SS composition to realize BS by considering the knowledge of how atomic BSs are mapped to existing SSs
    ● Non-functional requirements used in both phases to deal with variability in both business & IT level and select the best possible composition
Semi-automatic approach – requires user input to perform each phase but user interference is left to the minimum possible.

Deals with missing BS components.

Relies on existence of goal models, a domain model as well as descriptions of existing BS/SS.

- Goal models elicit functional business goals & capture functional variability in a business domain.
- BS/SS descriptions exploited to perform BS/SS composition.

State-of-the-art in service composition is advanced:

- All abstract service plans realizing a particular service are computed.
- Then the best plan as well as the best service combination concretizing the plan are selected according to non-functional requirements & plan selection criteria.
EXAMPLE – AEC DOMAIN
EXAMPLE – AEC DOMAIN

- **Design BS** mapping to **root goal** (Multi-Document Evaluation)
- **Input:** UserList, ExpertList, DesignDocInfo, RegInfo, EvaluationDocument
- **Output:** DesignDocument, RegDocList, SaveStatus, EvalDocList, EvaluationStatus, InformationStatus
- **BS Non-Functional Reqs:**
  - Cost < 4 euros
  - Security level >= “medium”
  - Reputation > 3 (max 5)
  - Business Domain Adequacy >= “medium”
GOAL-DRIVEN SERVICE IDENTIFICATION – 2ND APPROACH

- Semantic Meta-models exploited:
  - Goal ontology used for expressing Tropos goal models
    - Additional annotations for AND-decomposition to express the composition construct – useful for performance evaluation
  - Domain ontology in OWL
  - Functional BS/SS description based on OWL-S
    - Extended towards describing BSs and the goals partially/fully achieved
  - Non-functional BS/SS description based on OWL-Q
    - Extended towards distinguishing QoS vars at business & IT level, relationships between BS and SS non-functional specs & characterization of scope of constraints in a non-functional spec

- Incorporation of semantics:
  - Fixed vocabulary of terms
  - Automation of the method steps

CS 565 - LECTURE 8
GOAL-DRIVEN SERVICE IDENTIFICATION – 2\textsuperscript{ND} APPROACH
GOAL-DRIVEN SERVICE IDENTIFICATION – 2ND APPROACH

DESIGNER

getInput

[nextSol=no]

[isSat=no]

[missComp=no]

[modInput=no]

getNextInput

[nextSol=no]

[isSat=no]

[isLeaf=no]

BUSINESS SERVICE DESIGN METHOD

Business Service Composition

[nextSol=yes]

[isSat=yes]

[isLeaf=yes]

[missComp=yes]

[modInput=yes]

Business Service Realization

[nextSol=yes]

[isSat=no]

[isLeaf=no]
GOAL-DRIVEN SERVICE IDENTIFICATION – 2ND APPROACH
GOAL-DRIVEN SERVICE IDENTIFICATION – 2^ND APPROACH

- **Business Service Composition Production:**
  - Produce all *unique* (abstract) BS compositions
  - All BSs with same goal & I/O represented by *same BS task*
  - BS task also associated to all *unique BS task compositions* realizing it
  - BS task hierarchy is constructed in this way – requires appropriate *procedures* for BS publishing & updating to keep it *synchronized* as well as *expand* it
  - Also BS task subsumption hierarchy is maintained
  - BS composition *initiated* when new BS is *inserted* or when a *new BS* needs to be *designed*
GOAL-DRIVEN SERVICE IDENTIFICATION – 2ND APPROACH

BS Composition procedure:

1. Retrieve all BSs that are not exact matches of requested BS, produce one or more BS output (sub-)concepts and achieve a goal subsumed by the requested BS goal.

2. Examine for each BS retrieved which input are not provided by requested BS, which requested BS concepts are not produced by it & which remaining goals to be achieved to satisfy original goal.

3. If no BS achieves remaining goals or produces the missing output, then go to step 2 for next BS. Otherwise, call recursively the procedure with input the requested BS input & current BS output, with output the missing I/O concepts and goals the remaining goals to be achieved.

4. If no original output is missing & all goals have been covered, then store the BS composition produced if it is unique.
SS composition exploits almost the same procedure with the following differences:

- Goals are not considered
- Required BS I/O drives the procedure
- Needs interference as some SS compositions might be wrong/unrealistic
  - Could be remedied through the textual description of all goals – textual matching of goals
GOAL-DRIVEN SERVICE IDENTIFICATION – 2ND APPROACH

- Task-based Service Selection:
  - Create a Constraint Optimization Problem (COP) based on user’s non-functional requirements, produced unique/abstract service compositions, & the mapping of tasks to services and solve it
  - Multiple criteria to satisfy -> exploit Single Additive Weighting (SAW) technique to reduce to single-objective optimization problem -> production of one unique & optimal solution
  - Each criterion/non-functional parameter assessed via a utility function (applied on overall parameter value of the composition)
    - Utility in $[0.0, 1.0]$ plus allowance of minor violations of user requirements to cater for over-constrained user requests
    - Example:
      $$ uf_{\text{cost}}(c) = \begin{cases} 
      a + \frac{C-c}{C} \bullet (1-a), & c \leq C \\
      \max \left( a - \frac{c-C}{C} \bullet (1-a), a \right), & c > C 
      \end{cases} $$
GOAL-DRIVEN SERVICE IDENTIFICATION – 2ND APPROACH

- Task-based Service Selection:
  - **Weights** to criteria produced through the Analytic Hierarchy Process (AHP)
  - **4 criteria** for optimization:
    - *Part* – percentage of partially-realized BS components (wrt the number of leaf goals – mapping to atomic BSs)
    - *Num* – percentage of realized BS components
      - Preference over achievement of higher vs lower goals, less BS components realized, less complexity and less probability that BS is not automated
    - *Re-use* – percentage of BSs re-used
    - *S-deg* – matching degree of selected BS composition
    - *Cost*
    - *QoS*:
      \[
      qos = \sum_p w_p \bullet uf_p \left( v_p \right)
      \]
GOAL-DRIVEN SERVICE IDENTIFICATION – 2\textsuperscript{ND} APPROACH

- Task-based Service Selection:
  - 3 main decision variables:
    - $c_i$ – task combination is selected for service realization
    - $t_{ij}$ – task $j$ in combination is selected to participate in task combination $i$
    - $x_{jk}$ – service $k$ selected to realize task $j$
  - Differences with SS Selection:
    - SS exclusion constraints are considered
    - Part criterion not considered – non-existence of goals
    - Introduction of new criterion called loss: structural (non-semantic) information loss of selected concrete SS composition
    - Addition of new constraints to measure loss
maximize\(-w_{part} \cdot part + w_{cost} \cdot u_{f,cost} (cost) + \\
+ w_{qos} \cdot qos - w_{num} \cdot num + w_{re-use} \cdot re-use + \\
+ w_{s-deg} \cdot s-deg\)

\[ w_{part} + w_{cost} + w_{qos} + w_{num} + w_{re-use} + w_{s-deg} = 1 \]  

\[
\text{cost} = \sum_i c_i \cdot \sum_j t_{ij} \cdot \sum_k x_{jk} \cdot \text{cost}_{jk}
\]

\[
\text{num} = \sum_i c_i \cdot \sum_j t_{ij} 
\]

\[
\text{part} = \sum_i c_i \cdot \sum_j t_{ij} \cdot \sum_k x_{jk} \cdot \text{part}_{jk}
\]

\[
\text{qos} = \sum_p w_p \cdot u_{f,p} (v_p)
\]

\[
s-deg = \sum_i s-deg_i
\]

\[
s-deg_i = \begin{cases} 
1.0, & i \neq 0 \land \text{combination } i \text{ is exact match of required task} \\
0.5, & i \neq 0 \land \text{combination } i \text{ is plugin match of required task} \\
0.1, & i \neq 0 \land \text{combination } i \text{ is subsumes match of required task}
\end{cases}
\]

\[
s-deg_0 = \sum_k x_{0k} \cdot s-deg_{0k}
\]

\[
s-deg_{0k} = \begin{cases} 
1.0, & S_{0k} \text{ is exact match of } t_0 \\
0.5, & S_{0k} \text{ is plugin match of } t_0 \\
0.1, & S_{0k} \text{ is subsumes match of } t_0
\end{cases}
\]

\[
\text{part}_{jk} = \begin{cases} 
0, & t_j \text{ has goal } g \land \text{fullyAchieves } (S_{jk}, g) \\
1, & t_j \text{'s goal } g \text{ subsumes } S_{jk} \text{'s goal } g' \lor \text{partiallyAchieves } (S_{jk}, g)
\end{cases}
\]

\[
sol = \bigcup_{(i \text{ s.t. } c_i = 1)} \bigcup_{(j \text{ s.t. } t_{ij} = 1)} \bigcup_{(k \text{ s.t. } x_{jk} = 1)} S_{jk}
\]

\[
\text{unique} = \text{set}(sol)
\]

\[
\text{re-use} = \frac{\sum_{s \in \text{unique}} \text{isReused(sol, s)}}{\text{|unique|}}
\]

\[
\forall s, s' \in \text{sol}, \quad s \neq s' \land \text{achieves } (s, g) \land
\]

\[
\text{achieves } (s', g') \to \neg \text{excludes } (s, g, s', g')
\]

\[
v_p = \sum_i c_i \cdot v_{pi}, \forall p
\]

\[
v_{1i} = f_1 (v_{1il}), \forall i
\]

\[
v_{1il} = f_1 (\text{val}_{1j}), t_i \in ep^d, \forall i, l
\]

\[
v_{2i} = f_2 (\text{val}_{2j}), \forall i, j \text{ s.t. } c_i = 1 \land t_{ij} = 1
\]

\[
\text{val}_{p,j} = \sum_k x_{jk} \cdot qos_{pjk}, \forall p, j
\]

\[
\sum_i c_i = 1
\]

\[
\sum_k x_{jk} = 1, \forall j
\]

\[
\sum_p w_p = 1
\]
RECOMMENDED READING

- https://www.youtube.com/watch?v=L1tM0tMjdzy
- https://www.youtube.com/watch?v=ukU6TyXOMv0&t=36s
- https://www.youtube.com/watch?v=bPNfu0lZhoE