

Predicting Architectural Styles from Component Specifications

Sutirtha Bhattacharya

PTD Automation

Intel Corporation

Hillsboro, OR - 97124

sutirtha.bhattacharya@intel.com

Dewayne E. Perry

Empirical Software Engineering Lab (ESEL)

ECE, The University of Texas at Austin

Austin, TX 78712

perry@ece.utexas.edu

Background

⇒ Area of active Software Engineering Research

↳ Component Based Software Engineering (CBSE)

- Goal: Build software systems using pre-existing components thus reducing software costs and delivery time
- Research Focus: Understanding and resolving integration issues between the various components and establishing a common vocabulary for facilitating integration

↳ Architectural Styles

- Goals: Codify critical aspects about structure and constraints of complex software systems.
- Research Focus: Identification and documentation of styles with their associated quality attributes and facilitating their use

⇒ Gap

- ↳ No exploration/usage of architectural styles during component based software construction
- ↳ No mechanism for predicting quality attributes of a software system prior to its actual construction

Research Objective and Contribution

⇒ Research Objective

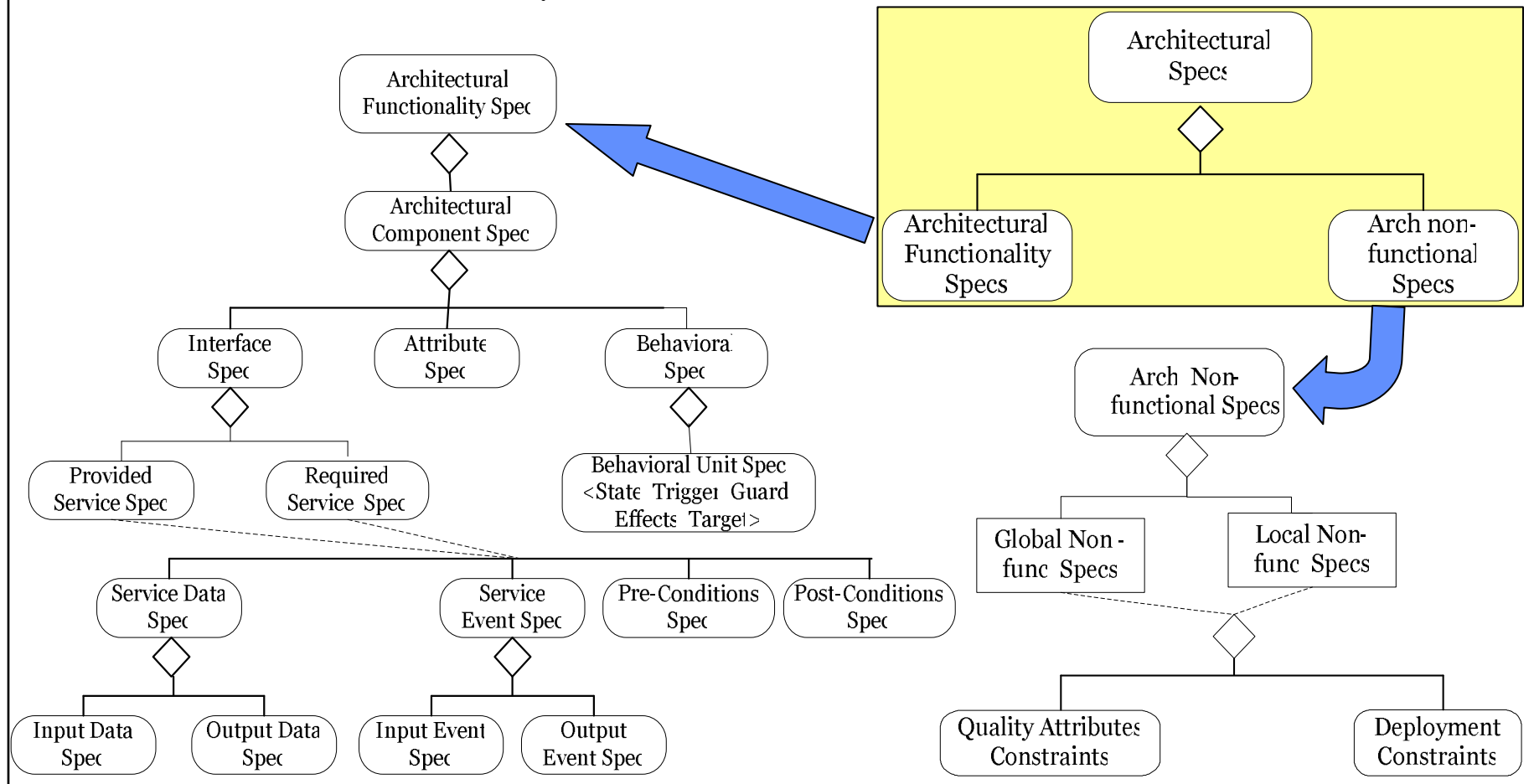
- ↳ Develop automated job-aids for software architects for selecting appropriate components which, when combined in a configuration, will satisfy the quality requirements set forth for the system
- ↳ Leverage large body of theoretical work in architecture styles for practical software construction

⇒ Contribution

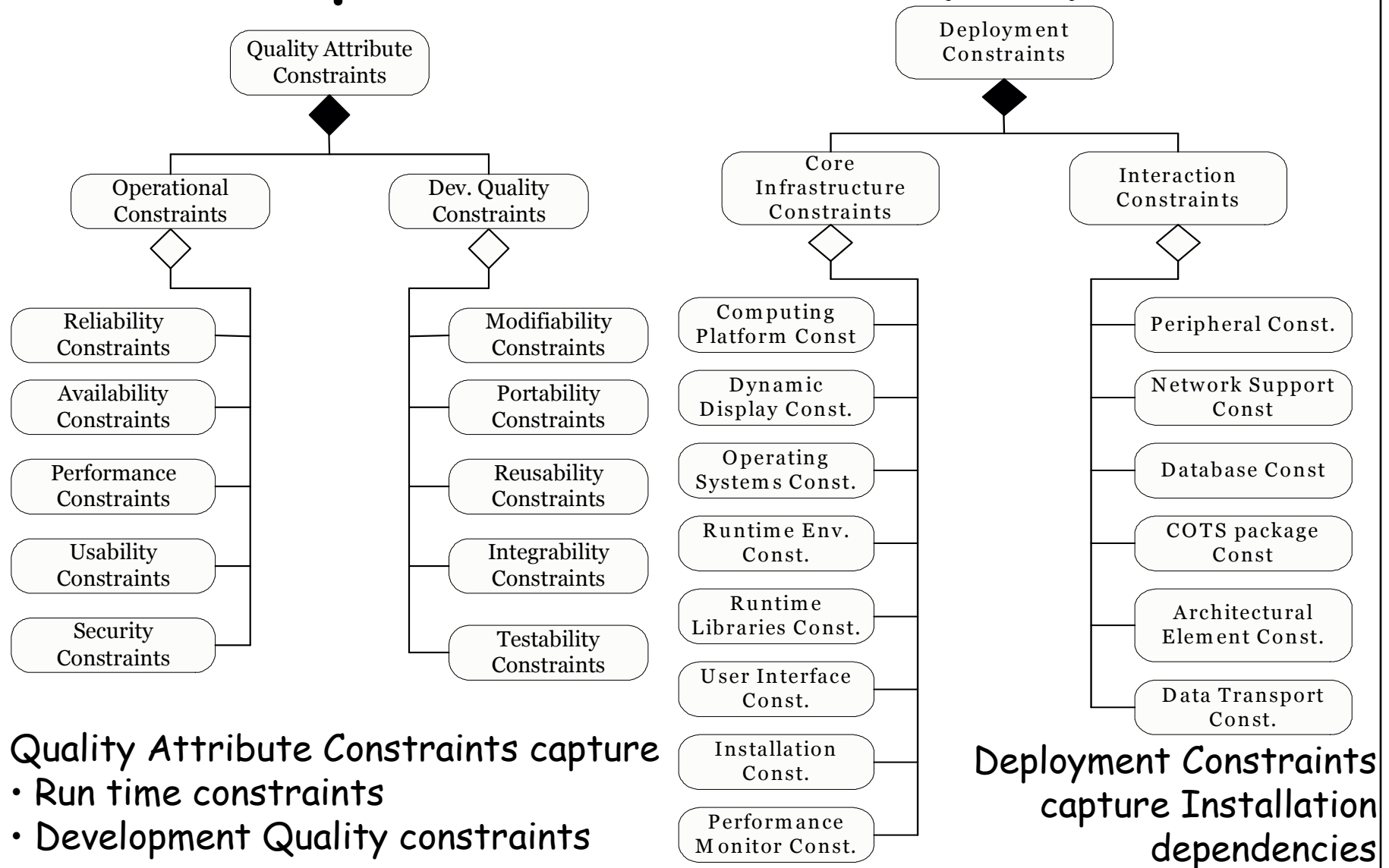
- ↳ A model for documenting software architectures
- ↳ Algorithms for predicting emergent architectural styles (hence quality attributes) during component composition

Specification Model (1/3)

- Software Architecture captured in terms of functional & non-functional specifications

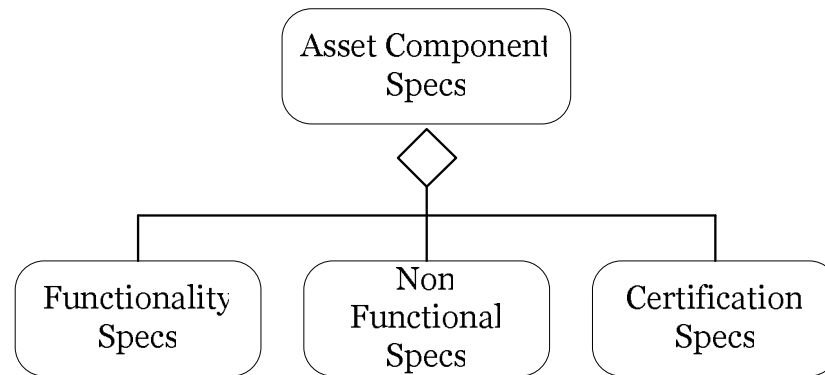


Specification Model (2/3)



Specification Model (3/3)

⇒ Asset Component Specifications



↳ Asset Functional and Non-functional Specs are captured using the same model as architectural components

⇒ Distinction between Architectural Component and Asset Component

↳ Architectural Components: Partitions the application domain - may not have actual realizations

↳ Asset Components: Components actually implemented (deployed as dlls, exe) - specified in terms of architectural components

Predicting Architectural Styles - Background

- ⇒ Based on Shaw and Clements work on feature-based classification of architectural styles [3]
- ⇒ Shaw and Clements proposed that architectural styles can be discriminated among each other by focusing on the following feature categories.
 - ↳ Constituent Parts i.e. the components and connectors
 - ↳ Control Issues i.e. the flow of control among components
 - ↳ Data Issues i.e. details on how data is processed
 - ↳ Control/Data Interaction i.e. the relation between control and data
 - ↳ Type of Reasoning: Analysis techniques applicable to the style

Predicting Architectural Styles - Assumptions

⇒ Assumptions

- ↪ There exists a component repository in which components have been specified using our asset specification model
- ↪ A System Integrator identifies a deployment use-case (made up of a list of services) that needs to be satisfied
- ↪ For identifying the configuration of components for satisfying the use case, the System Integrator queries the repository for the available components
- ↪ The proposed architectural style related reasoning will be done on the set of components returned by the component repository based on the system integrator's query

⇒ Reasoning Capability will support

- ↪ Determination of whether the set of components returned by the repository conform to any specific architectural style
- ↪ Identification of a set of components that conform to a desired architectural style and hence support the desired set of quality attributes.

Algorithm for predicting Architectural Styles (1/3)

- ⇒ Step 1: The System Integrator identifies a use case/scenario for which a software configuration needs to be built
- ⇒ *Step 2*: For each service in the use case, we identify the best fit candidate from the component repository i.e. the component with the highest value of the Service Compliance Coefficient [6] and build the *Base Component List*.
 - ↳ Service Compliance Coefficient: Weighted average of the Input and Output Data Compliance Coefficient, the Input and Output Event Compliance coefficient and the Pre and Post Condition Compliance Coefficient [See backup for details]
- ⇒ *Step 3*: For each component in the *Base Component List*, we make a note of its *Component Type Attribute*. If all the components are not of the same type, we consider the component type of the set of components to be the one that is most common.
 - ↳ *Component Type Attribute is associated with the Asset Component Specifications*
- ⇒ *Step 4*: For each component in the *Base Component List*, we make a note of the *Connector Type* attribute in the Data Transport Spec. If all the connectors are not of the same type, we consider the connector type of the configuration of components to be the one that is most common.
 - ↳ *Connector Type attribute is captured in the Interaction Constraints of the Deployment Constraints Specifications*

Algorithm for predicting Architectural Styles (2/3)

- ⇒ *Step 5:* Determine the Control Topology of the set of components the Control Flow List
 - ↳ The Control Flow list is an ordered list of components needed for satisfying all the services in the use case as well as the list of components that generate the input and output events associated with each service in the use case
- ⇒ *Step 6:* Determine the Control Synchronicity of the configuration of the components
 - ↳ Based on the generation and consumption of events from the service execution
- ⇒ *Step 7:* The Data Topology of the configuration of components is determined by developing the Data Flow List
 - ↳ The Data Flow list is an ordered list of components driven by the data consumed (input data) and the data produced (output data) by the services in the use case.
- ⇒ *Step 8:* The Data Continuity of the configuration is determined
 - ↳ *Continuous or sporadic based on data utilization pattern*
- ⇒ *Step 9:* Determine whether the Control and Data Topologies are isomorphic
- ⇒ *Step 10:* From the feature category attributes derived in Steps 3 to Step 10, we reference the next foil to determine the Architectural Style of the set of components.

Algorithm for predicting Architectural Styles (3/3)

Style	Constituent Parts		Control Issues		Data Issues		Control/Data Interaction
	<i>Components</i>	<i>Connectors</i>	<i>Topology</i>	<i>Synchronicity</i>	<i>Topology</i>	<i>Continuity</i>	<i>Isomorphic Shapes</i>
Data Flow Architectural Styles							
Batch Sequential	Stand-alone programs	Batch data	Linear	Sequential	Linear	Sporadic	Yes
Data-Flow Network	Transducers	Data Stream	Arbitrary	Asynchronous	Arbitrary	Cont.	Yes
Pipes and Filter	Transducers	Data Stream	Linear	Asynchronous	Linear	Cont.	Yes
Call and Return							
Main Program/Subroutines	Procedure	Procedure Calls	Hierarchical	Sequential	Arbitrary	Sporadic	No
Abstract Data Types	Managers	Static Calls	Arbitrary	Sequential	Arbitrary	Sporadic	Yes
Objects	Managers	Dynamic Calls	Arbitrary	Sequential	Arbitrary	Sporadic	Yes
Call based Client Server	Programs	Calls or RPC	Star	Synchronous	Star	Sporadic	Yes
Layered	-	-	Hierarchical	Any	Hierarch.	Sporadic	Often
Independent Components							
Event Systems	Processes	Signals	Arbitrary	Asynchronous	Arbitrary	Sporadic	Yes
Communicating Processes	Processes	Message Protocols	Arbitrary	Any but Sequential	Arbitrary	Sporadic	Possibly
Data Centered							
Repository	Memory, Computations	Queries	Star	Asynchronous	Star	Sporadic	Possibly
Blackboard	Memory, Components	Direct Access	Star	Asynchronous	Star	Sporadic	No

Conformance Confidence Index

- ⇒ Measures the degree of conformance of a component configuration to a given architectural style
 - ↳ Higher the value of CCI, the more compliant is the configuration to the corresponding architectural style.

$$CCI = \sum_{fc \in FCA(s)} \frac{w_{fc} \times V_{fc}}{|FCA(s)|}$$

↳ Legend

- $FCA(s)$: The set of feature category attributes relevant for a given style s . In our case $FCA(s) = [\text{Components, Connectors, Control Topology, Synchronicity, Data Topology, Data Continuity, Isomorphic Shapes}]$ for all styles by the Shaw Clements classification.
- w_{fc} = the weight of the feature category attribute in the determination of the style. This factor can be ignored if empirical analysis shows that all the feature category attributes have equal weighting. If they are found relevant (as likely they will be), the values have to be determined individually for each style
- $V_{fc} = 1$ if our approach reveals that the corresponding feature category for a configuration matches the Shaw Clements classification for the given style, 0 otherwise
- S = style under consideration

References

- [1] Perry, D. E., Wolf, A. L., "Foundations for the Study of Software Architectures", ACM Software Engineering Notes, 17, 4, October 1992, 40-52
- [2] Abowd, G., Allen, R., Garlan, G., "Using style to understand descriptions of software architecture", Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering, 1993, 9-20
- [3] Shaw, M., Clements, P., "A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems", Proceedings of the 21st International Computer Software and Applications Conference, 1997, 6-13
- [4] Habermann, A. N., Perry, D. E., "Well Formed System Composition. Carnegie-Mellon University, Technical Report CMU-CS-80-117. March 1980
- [5] Bhattacharya, S. "Specification and Evaluation of Technology Components to Enhance Reuse," Masters Thesis, The University of Texas at Austin, July 2000
- [6] Bhattacharya, S., Perry, D. E., "Contextual Reusability Metrics for Event-Based Architectures", The 4th International Symposium on Experimental Software Engineering, November 2005, Australia
- [7] Perry, D. E., Wolf, A. L., "Software Architecture", August 1989.
<http://www.ece.utexas.edu/~perry/work/papers/swa89.pdf>

Backup

Calculating Service Compliance Coefficient(1/2)

⇒ Input and Output data Compliance Coefficient

$$\text{IDCoeff}(s) = \frac{1}{|\text{IDEn}(s)|} \sum_{en \in \text{IDEn}(s)} \frac{|\text{IDEI}_{\text{regd}}(s, en)|}{|\text{IDEI}(s, en)|}$$

$$\text{ODCoeff}(s) = \frac{1}{|\text{ODEn}(s)|} \sum_{en \in \text{ODEn}(s)} \frac{|\text{ODEI}_{\text{regd}}(s, en)|}{|\text{ODEI}(s, en)|}$$

⇒ Input and Output Event Compliance Coefficient

$$\text{IECoeff}(s) = \frac{|\text{IE}_{\text{regd}}(s)|}{|\text{IE}(s)|} \quad \text{OECoeff}(s) = \frac{|\text{OE}_{\text{regd}}(s)|}{|\text{OE}(s)|}$$

⇒ Pre and Post Condition Compliance Coefficient

$$\text{PreCondCoeff}(s) = \frac{|\text{PreCond}_{\text{regd}}(s)|}{|\text{PreCond}(s)|} \quad \text{PostCondCoeff}(s) = \frac{|\text{PostCond}_{\text{regd}}(s)|}{|\text{PostCond}(s)|}$$

Calculating Service Compliance Coefficient(2/2)

⇒ Service Compliance Coefficient

$$SvCoeff(s) = \frac{[IDDep(s) \times IDCoeff(s) + ODDep(s) \times ODCoeff(s) + IEDep(s) \times IECoeff(s) + OEDep(s) \times OECoeff(s) + PreCondDep(s) \times PreCondCoeff(s) + PostCondDep(s) \times PostCondCoeff(s)]}{IDDep(s) + ODDep(s) + IEDep(s) + OEDep(s) + PreCondDep(s) + PostCondDep(s)}$$

⇒ Legend

- ↔ $IDEn(s)/ ODEn(s)$: Set of Input/Output Data Entities for service s .
- ↔ $IDEIregd(s, en)/ ODEIregd(s, en)$: Set of Input/Output Data Elements for the entity en of service s , to which the component is registered.
- ↔ $IDEI(s, en)/ ODEI(s, en)$: Set of Input/Output Data Elements for the entity en of service s .
- ↔ en : An entity belonging to the set $IDEn(s)/ ODEn(s)$
- ↔ $IE_{regd}(s)/ OE_{regd}(s)$: Set of Input/Output Events for service s , to which component is registered.
- ↔ $IE(s)/ OE(s)$: Set of Input/Output Events for service s
- ↔ $PreCondregd(s)/ PostCondregd(s)$: Set of Pre/Post Conditions for service s , to which component is registered.
- ↔ $PreCond(s)/ PostCond(s)$: Set of Pre/Post Conditions for service s .
- ↔ $IDDep(s)/ ODDep(s)$: Total number of services generating /consuming the input/output data entities required by service s .
- ↔ $IEDep(s)/ OEDep(s)$: Total number of services that generate/depend on trigger events of /from the service s .
- ↔ $PreCondDep(s)/ PostCondDep(s)$: Total number of services responsible for the set of pre/post conditions