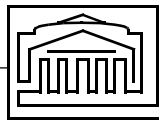


IN MODEL

in the

ODP ARCHITECTURAL FRAMEWORK

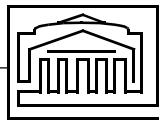
(A Long-Term IN Architecture View)

"Integrating Telecommunications and Distributed Computing Technology"

Kazi Farooqui

Department Of Computer Science,
University of Ottawa,
Ottawa, Canada.

(Presented at IEEE IN-95 Workshop,
Ottawa, May 1995)



1. Integrating Telecommunications and ODP

❑ IN Long-Term Architecture:

Synergy of:

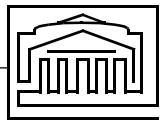
1. IN Conceptual Model (Higher-Order Capability Sets)

+

2. Distributed Computing Architectures such as RM-ODP.

❑ Why Distributed Computing Technologies:

-Introduction of Advanced Telecommunication Services makes the Telecommunication Platform a large "Distributed Processing System".



❑ IN Model: Breakthrough in Telecom Industry

- Separation of *Service Control* from *Switching Control*

- A first step in separating a "*service architecture*" from "*network architecture*".

Still a Primitive Model.....

- A Functional Model of Building Blocks.

- Defines Low-Level Building Blocks.

- No definition of "distribution-support" components.

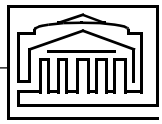
- Talks of Data Attributes.

- Communication structures between building blocks not clearly defined.

- Not suitable for supporting advanced telecommunication services with "*distributed processing*" requirements.

- Lack of guidelines for separating "Architecture" from "Protocols" and "Applications".

❑ IN-Model: Needs evolution from "A Functional Model" to a "*Distributed Object Model*" embracing distributed processing concepts and mechanisms.



- ODP permits modelling of different ***aspects*** of "IN-Service" and "IN-Platform" in a consistent framework:

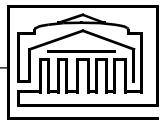
  1. Enterprise: Objectives, Requirements, and Policy of Service from its users point of view. Who are the users and providers of service and what are their roles.

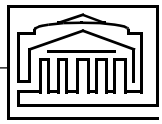
  2. Information: Service Elements, Information Flow between Service Elements, Relationships between Service elements, Semantics of Information processed and exchanged.

  3. Computation: *Distribution-transparent* and IN-Platform independent "Service Support" in terms of service components and their interactions.

  4. Engineering: Service-Specific QoS Support,
     Distribution Transparency Support,
     Communication Support,
     Switching/Transmission Support.

  5. Technology: How to implement and install services and platforms.





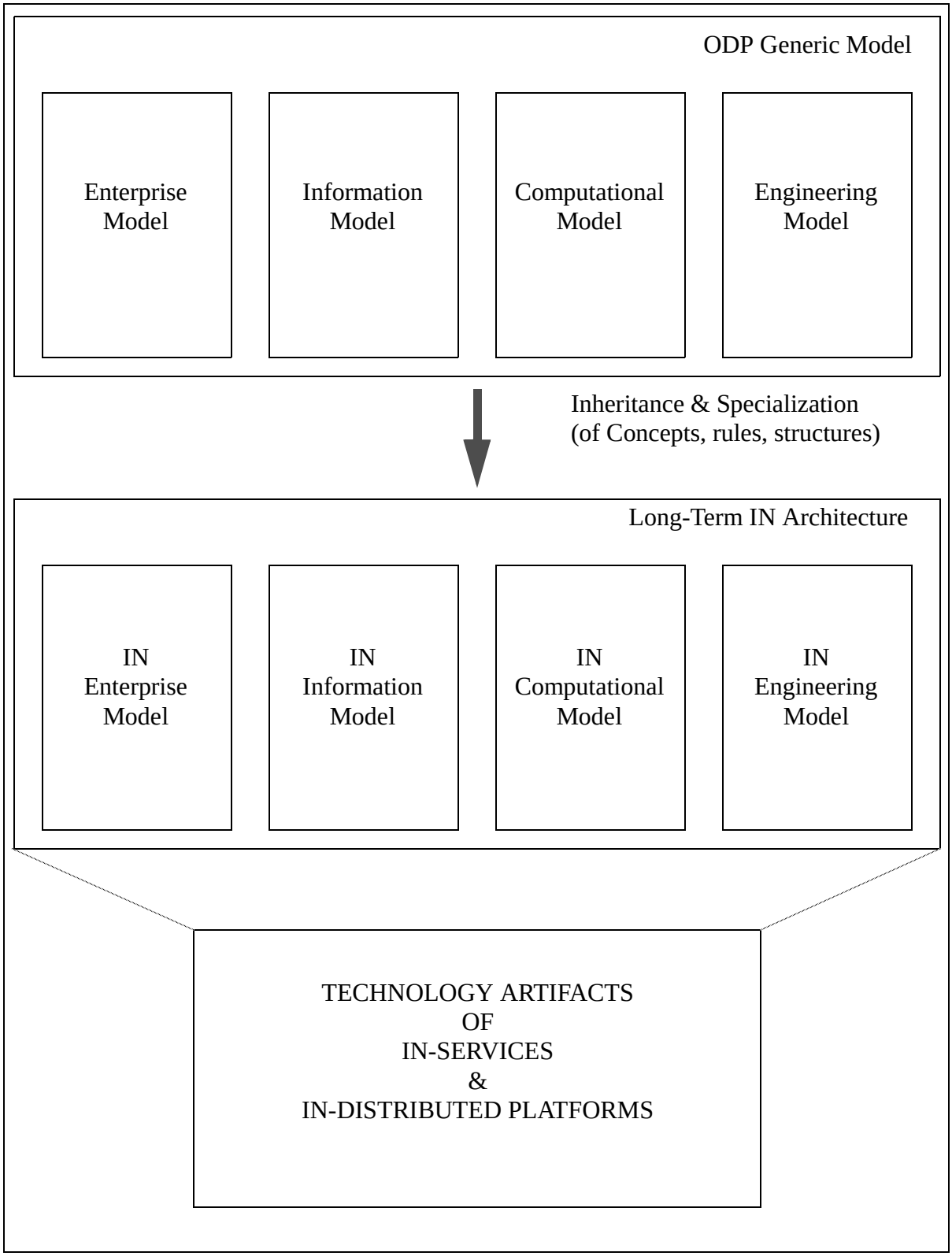


Figure 1. ODP Modelling of IN Systems

❑ Motivation for ODP Technology:

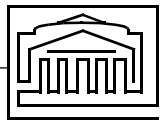
1. To serve as a "Distributed Object Model" for service architecture and distributed platform architecture.

2. To provide abstract object modelling concepts such as object, interface, template, type, class, etc. required for object-based distributed systems.

3. To design an "Object-Based IN-Distributed Processing Platform".

4. To provide an advanced "Service Programming Environment" on top of the IN-Distributed Platform. (To make IN-Structured-Network a "Programmable Entity").

5. ODP Model promotes interoperability, evolvability, scalability, software reuse, support for new services, independence from technology.



2. ODP Computational Model: What Does it Offer ?

❑ What does ODP Computational Model offer to IN Model:

1. Generic Object-Oriented Interaction Model:

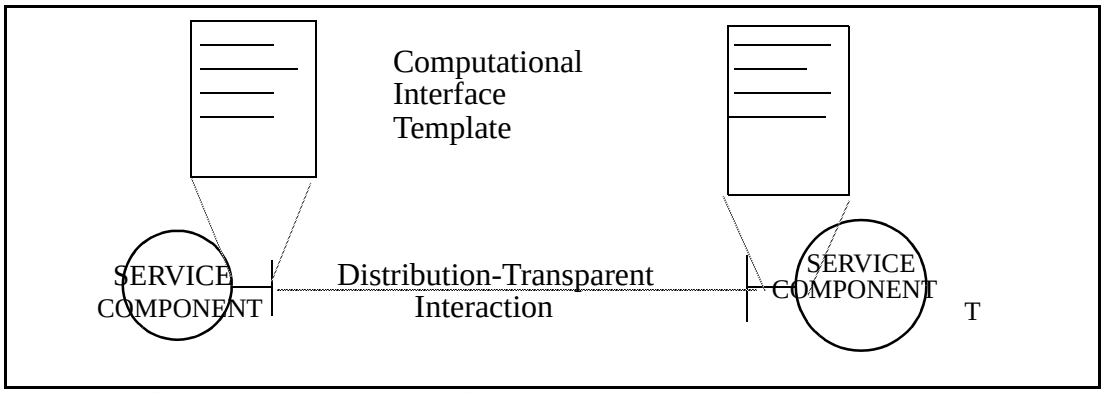
2. Binding Model:

3. Type Model:

4. Distributed Programming Model:

5. Trading Model:

1. Generic Object-Oriented Interaction Model:



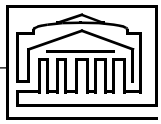
Objects Interact by Operation Invocations OR Stream Flows at their Interfaces.

- IN-Service is an object-based distributed application.

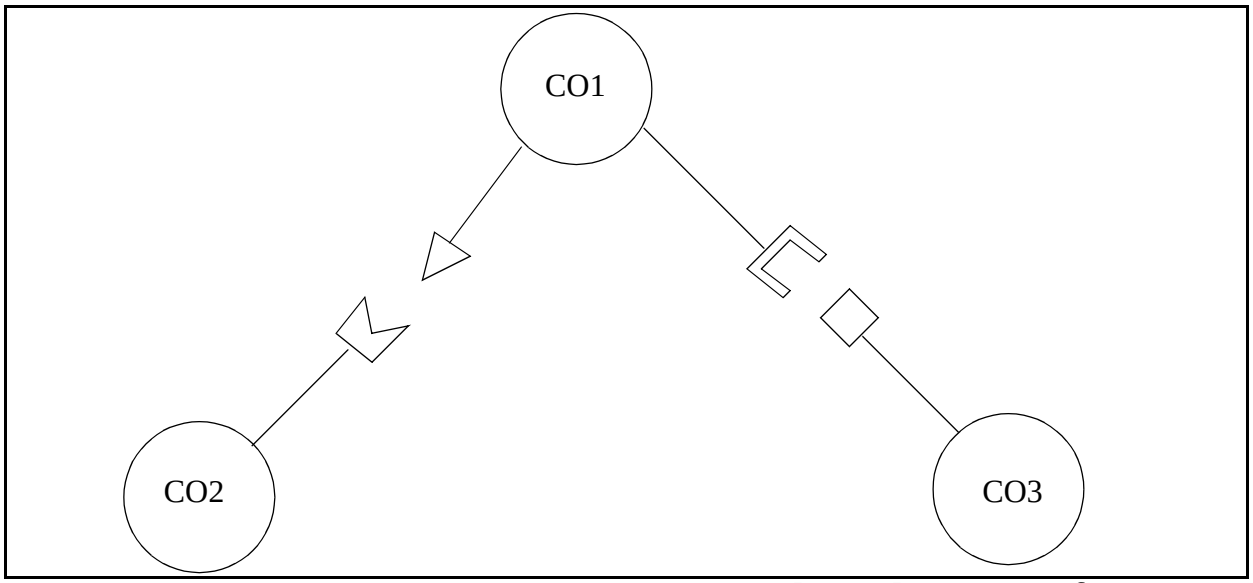
- Modelling support for different Interaction Structures and Information Types:
  a. *Interrogations*, *Announcements*.
  b. *Operational Interfaces*, *Stream Interfaces*.

- IN-Service Components are *distributed computational objects* interacting with each other by sending *operation invocations* or continuous *stream flows* at their *interfaces*.

- *Distribution-Transparency*: Support for *remote interactions* with *concurrent*, *replicated* and *mobile* objects in *heterogeneous telecom environments*.



2. Type Model:



INTERACTION MODEL: Interactions based in interface type matching

- IN-Services are complex applications which can be created out of available components.

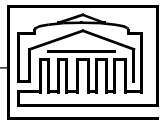
- A repository of components and their *interface types* enables faster service creation.

- Service creation/composition based upon *type-checked* (late) binding of (interfaces of) service components.

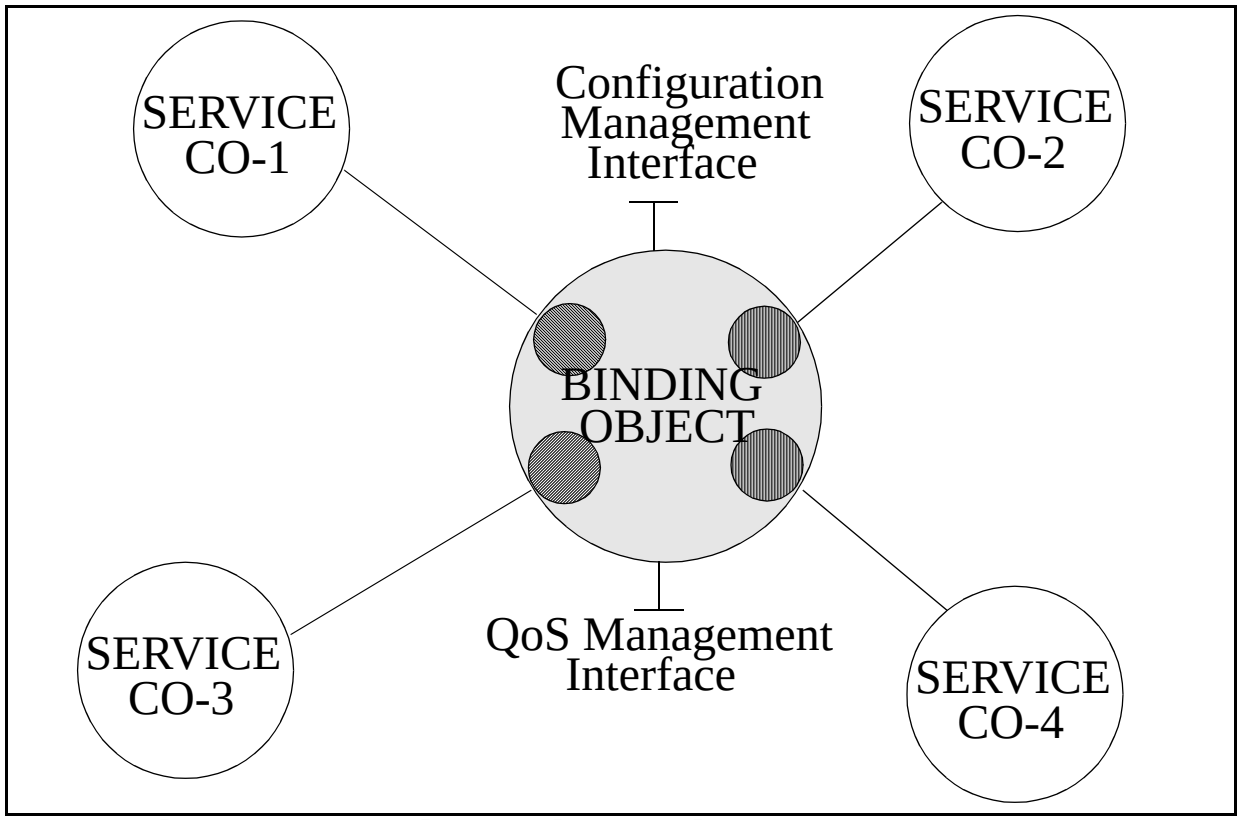
- *substitutability* of one service (component) by another.

- An asset for "Service-Creation-Environments".

- Current ODP Type Model: Based on *operation signature compatibility*. Subtyping can be extended to *QoS compatibility* and *behavior compatibility* to meet the needs of IN Services.



3. Binding Model:



Multi-Party / Multi-Media Binding Support Structures

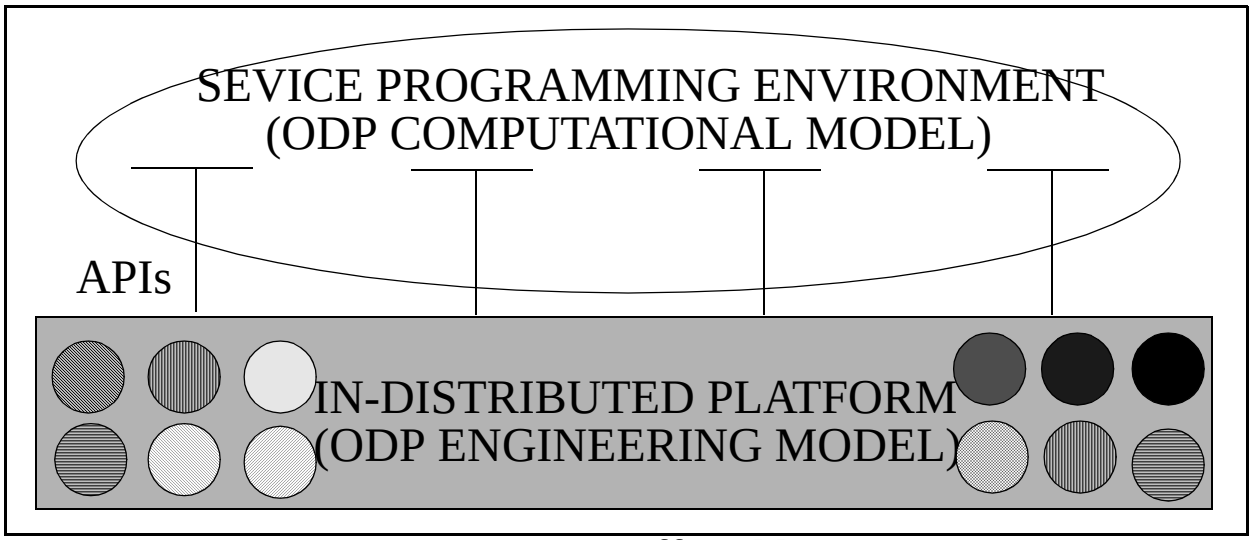
- Support for *primitive binding models* (interrogations, announcements), and *complex binding models*.

- *Binding Objects* enable complex binding structures to be supported between IN-Service components.

- Services (Applications) have control on *configuration* and *quality of service* of the complex binding.

- Support for *multiparty* and *multi-media* binding.

4. Distributed Programming Model:

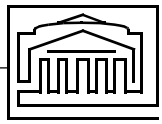


Service Programming: APIs offered by IN-PLATFORM

- *Abstract* object-based language-independent programming model *concretized* for IN-Service Programming.

- *IN-Service Environment*: A large programming environment capable of building and executing service applications on the supporting IN-Platform (A large virtual machine).

- *IN-Service Logic*: Definition of Application Programming Interfaces for composing service components.



5. Trading Model:

- Future IN Services to evolve into a "Service-Market".

- Different *types* of IN Services of different *quality properties* and *costs* advertised through "Service-Traders".

- Discovery and availability of service is a key factor in the success of a "*Service Architecture*".

- Also useful in a "Service Creation Environment".

2. ODP Engineering Model: What Does it Offer ?

❑ What does ODP Engineering Model offer to IN Model:

1. Model of an Object-Based Distributed Platform:
   - Concepts, Rules, Structuring Framework (of nodes, capsules, clusters, objects) for the organization of an IN-Distributed Platform that supports execution and interaction of IN Services.

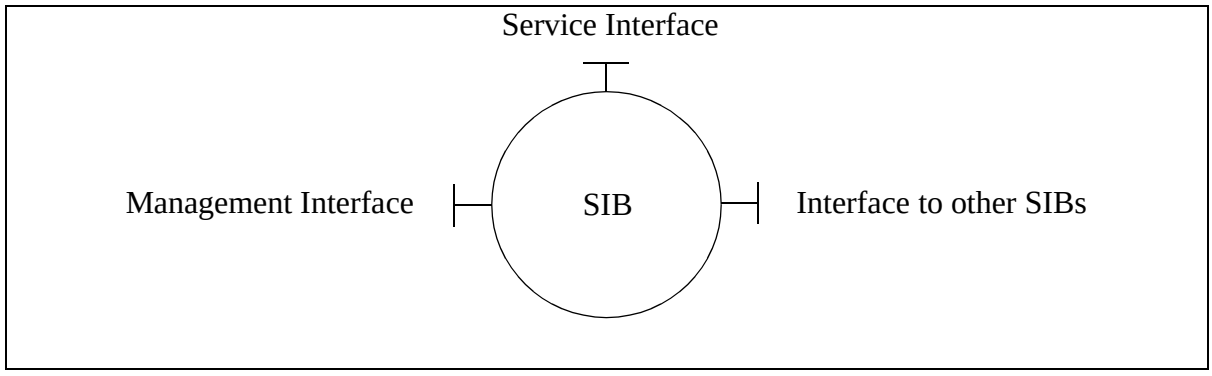
2. Distribution Transparency Support Mechanisms:
   - An engineering framework of "transparency support mechanisms" for the provision of l*ocation transparency*, *migration transparency*, *transaction transparency*, *group transparency* etc. to the interactions between IN-Services (components).

3. QoS Support Framework:
   - An engineering framework of "QoS support mechanisms" for the provision of QoS properties such as throughput, delay, jitter, media synchronization, etc.

4. Service Machine:
   - The ODP concept of "*channel* between distributed engineering objects" can be specialized as a *service machine* that supports the service-specific requirements of *distribution transparency, communication*, and other *QoS*.
   - Engineering representation of service execution and interaction support.
   - Dynamically *configurable* and *tailored* to individual service requirements.

3. Computational Model of IN

- IN Lacks a well-defined Computational Model.
- Service Plane/Global Functional Plane serves as a primitive Computational Model of IN.
- Service Independent Building Block: Network-wide, service independent, technology-independent service building capability.
- Computational Modelling of SIB:
  - Generalized as an *object* with *multiple interfaces* and *internal concurrency*.
  - SIBs have s*ervice interface*, *management interface*, and *interface to other SIBs*.
  - Modelling of services and its management at the same level.
  - Every object is responsible for its own management. Management interfaces are used by *managing applications* that manage the object. Standardization of management interface simplifies management of products.

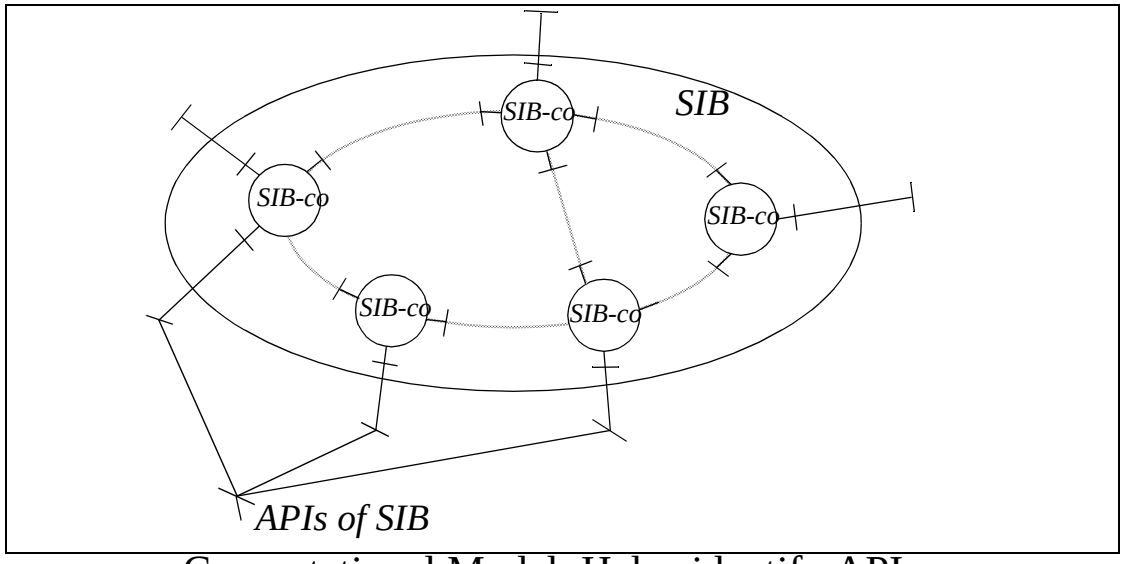


SIB: A Computational Object with multiple interfaces

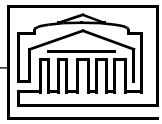
- SIB *<u>functionality is distributed</u>* over the network (platform). "Computational modelling" of SIBs allows to identify:
  a. SIB components (computational objects) which are candidates for distribution, migration, activation, deactivation, and management.
  b. Interfaces of SIB components. Some of these interfaces become <u>APIs</u>.
  c. Distribution-transparent interactions between component interfaces.
  d. Environment constraints applicable to components and their interactions. Enables to express QoS such as throughput, delay, jitter, transparency requirements, etc.

- "Interface definitions" make service components independent of the environment and can be reused in any environment.

- Different interfaces of the SIBs (or service components) at the GFP are supported by different protocols at DFP. No need to identify protocols at the GFP level.



<u>Computational Model: Helps identify APIs</u>.



❑ Interaction between SIBs:

- Operational and Stream Interfaces.

- Interrogations: Operation invocations with multiple terminations (results).

- Announcement: Invocation with no terminations.

❑ Global Service Logic:

- Computational Modelling of Distributed Application (IN Service).

- Composition of concurrent (and distributed) service components, SIBs (Computational Objects) interacting by operation invocations/stream flows at their interfaces.

- A Programmable entity.

- IN CS-1 models *sequential* invocation of SIBs.

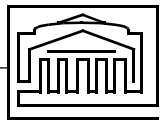
vs.

ODP Computational Model allows *concurrent* execution of SIBs. (*Concurrency* is separated from *communication*).

❑ Service Creation at GFP:

SIBs: *Chaining* Principle

ODP Computational Objects: *Composition* (Reference, Part of Principle, Recursion) Principle.
Inheritance Principle.



❑ Basic Call Process:

1. Point-to-Point telephony concepts should give way to more general concepts of *service session*. BCP evolves into a "*Generic Session Manager*". (No need to associate a service with a call model).

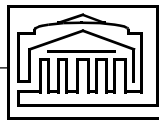
2. "*ODP Group*" concept provides a neat architectural solution for modelling "*Generic Session Manager*".

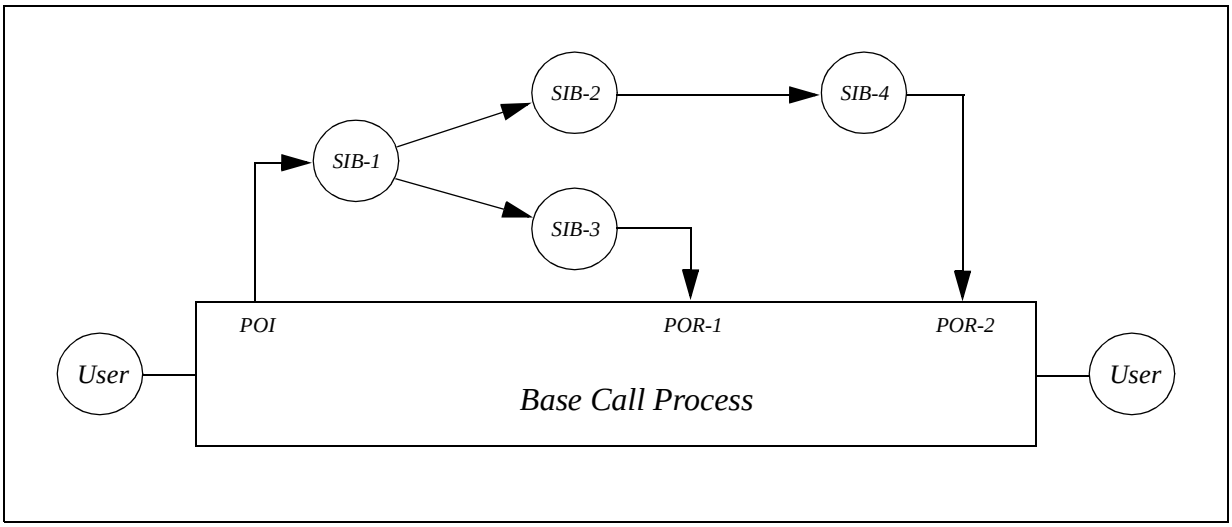
3. Generic Session Manager:

- Hides details of call/connection management.
- Session-Based rather than Switch-Based.
- Support for Multiparty and Multimedia Binding.
- Support for multiple points of control.
- Support for mobility of end-parties.
- scalable solution for including other objects such as feature interaction managers, session managers, etc.
- end users can suspend and resume logical sessions.

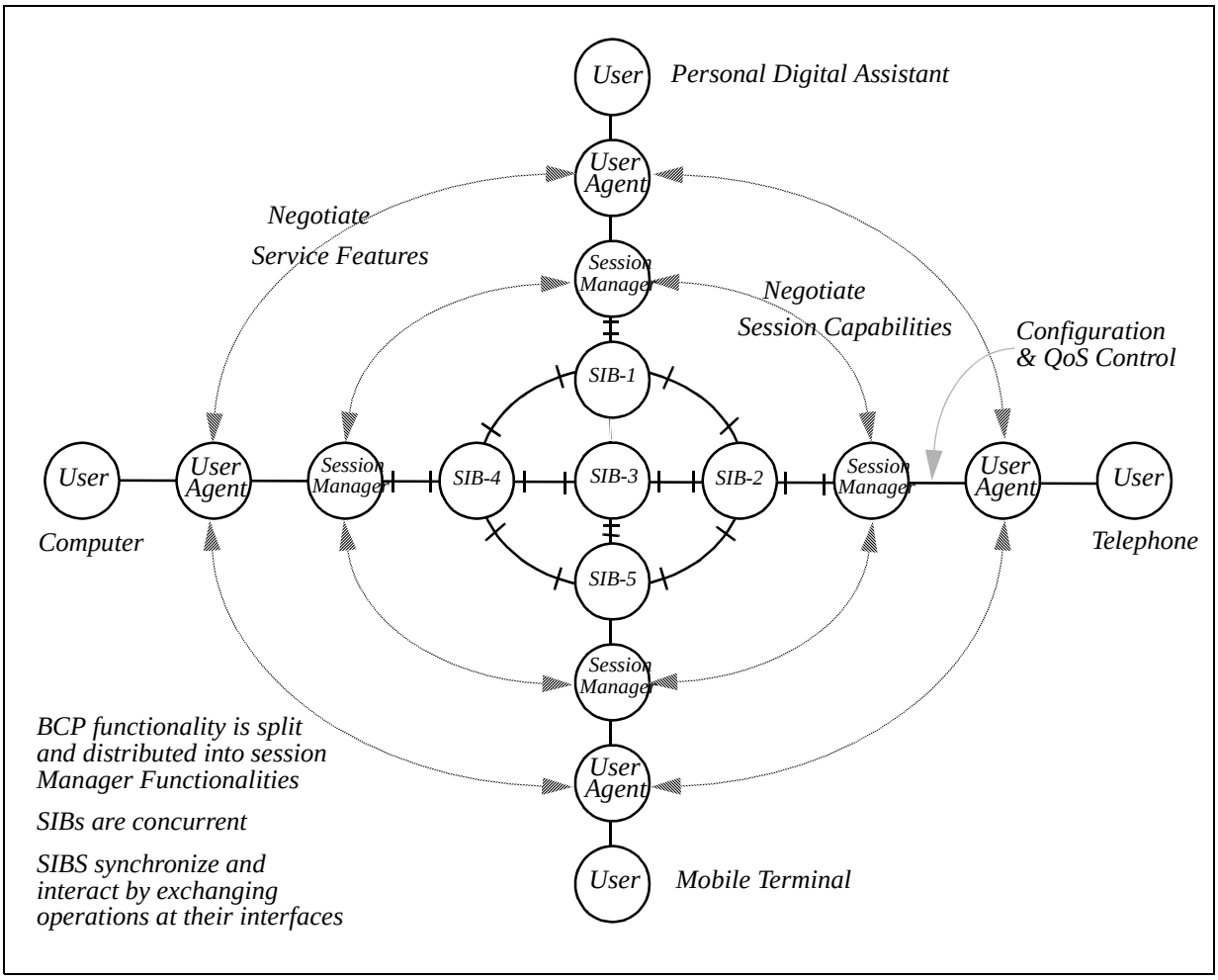
4. Generic Session Manager Configuration:

- Users (in a session) are attached to an "*agent*".
- Agents are driven by user "*policies*".
- Agents perform "*negotiation*" based upon policies.
- User control of "session configuration" and QoS.

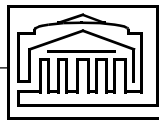




Single Point of Control can be Generalized to multiple-points of Control



Generalized Call Model



4. Enginering Model of IN

- IN lacks a well defined Engineering Model.
- Distributed Functional Plane serves as a primitive Engineering Model.
- Engineering Model describes the organization of the IN-Distributed Platform as a set of *nodes* interconnected by communication network.
- Engineering Model describes how to deploy SIBs/Service Components in order execute them and to enable interaction between them.
- Service Components could be information processing entities or data (container) entities (multimedia data).
- Service components are represented as *Basic Engineering Objects* (BEOs) which are distributed entities.
- A set of service-related BEOs (EFs) at a given node are organized as a *cluster* (FEA). Clusters models service packages which are deployable as a unit.
- Information processing service components are located in Service Control Function (SCF) *capsules*.
- Data storage service components are located in Service Data Function (SDF) *capsules*.

- A *node* may contain SCF, SDF, SRF, SSF/CCF *capsules.*

- Service components on distributed nodes interact to provide *distributed service logic*.

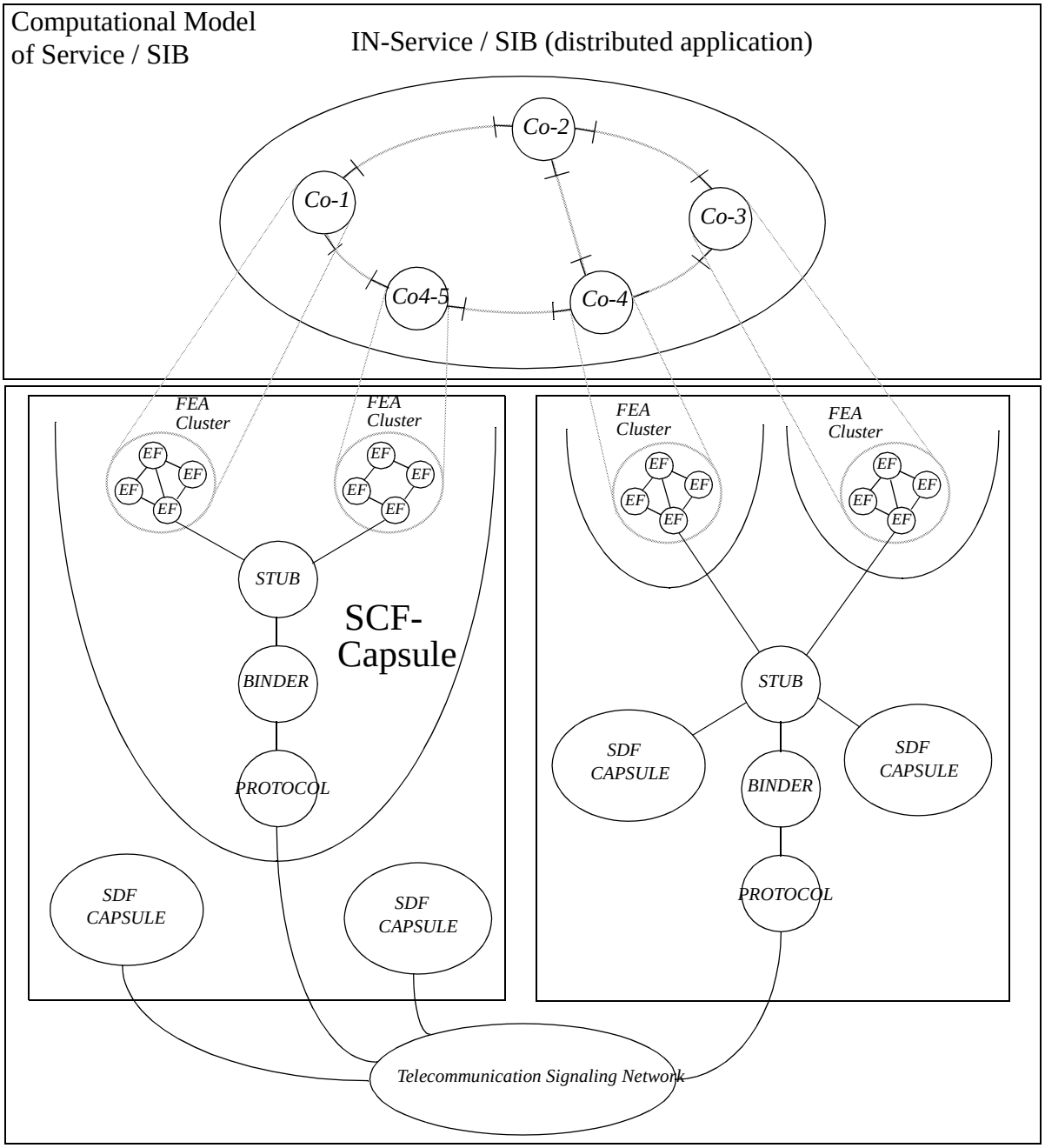
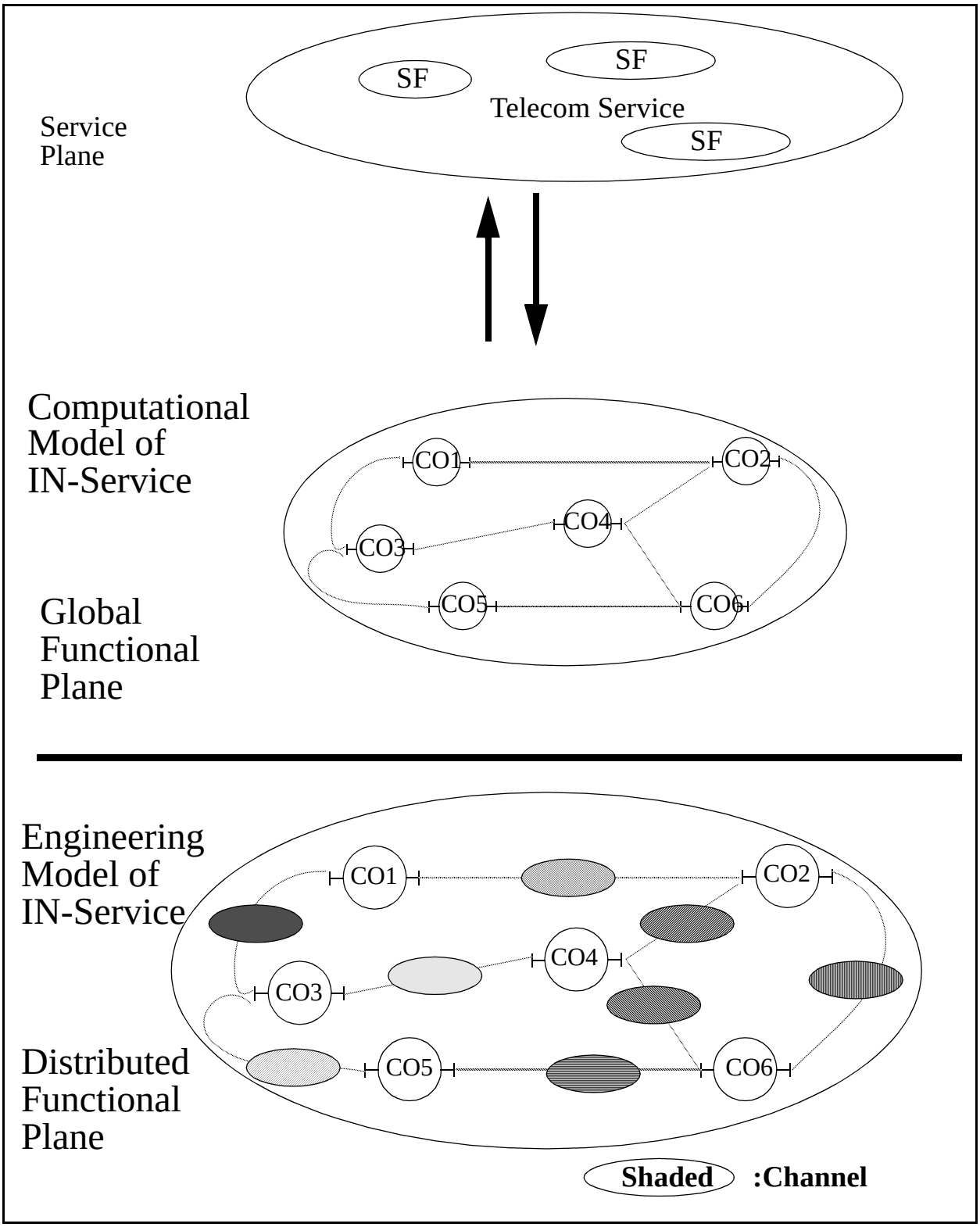


Figure 5. Enhanced Engineering Model of DFP

BINDER: Location Transparency Binder
Migration Transparency Binder
Group Transparency Binder
Failure Transparency Binder
Resource Transparency Binder

STUB: Transaction Function

PROTOCOL: Any Suitable Protocol between:
SCF - SCF
SCF - SDF
SCF - SRF
...



Computational and Engineering Models of IN-Service

❑ Functional Entities in the IN Engineering Model:

1. Limited Scope of Function Definitions: SCF, SDF, SRF are service-specific functions while SSF/CCF is a communication function.

2. IN Distributed Platform (DFP) needs generic ODP "*distribution-support*" functions, such as:

   a. Service TRANSACTION FUNCTION (STF).

   b. Service REPLICAION FUNCTION (SReF).

   c. Event Notification Function (ENF).

   d. Service GROUP FUNCTION (SGF).

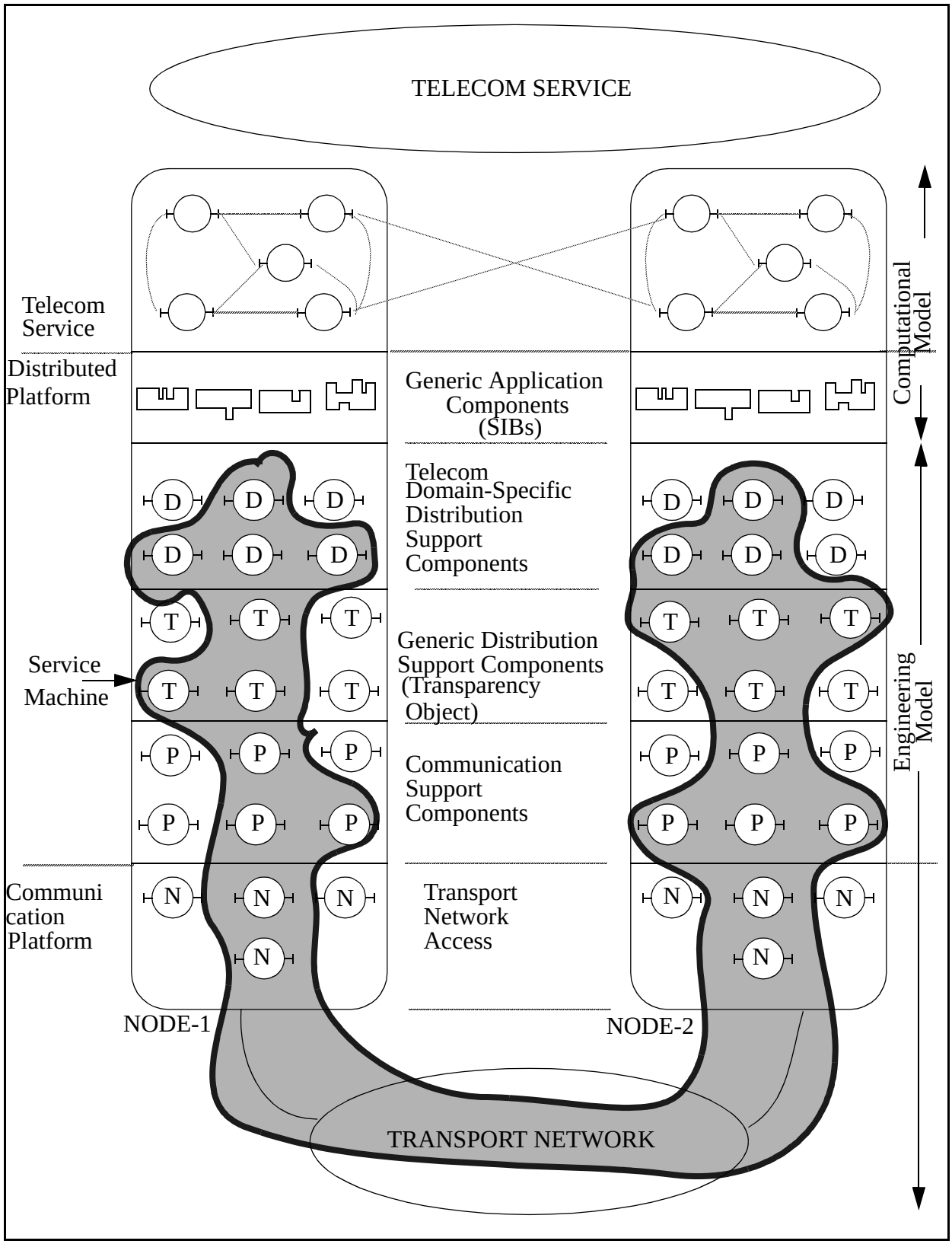
   e. Service MIGRATION FUNCTION (SMiF).

   f. Service RELOCATION FUNCTION (SReF).

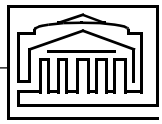
   g. Service TRADING FUNCTION (STF)

   h. Service SECURITY FUNCTION (SSeF).

   i. Other Domain-Specific Distribution-Support Functions.



Service Machine: Customization of Engineering Model.



❑ ODP DISTRIBUTION TRANSPARENCY:

Services may be designed and interactions between their components can take place, independent of where within the system they are located, migrated, replicated, or how they are organized. The details of local access, remote access, or concurrent access are hidden.

1. Location Transparency: PCS/UPT Applications and UMTS Platforms, etc.

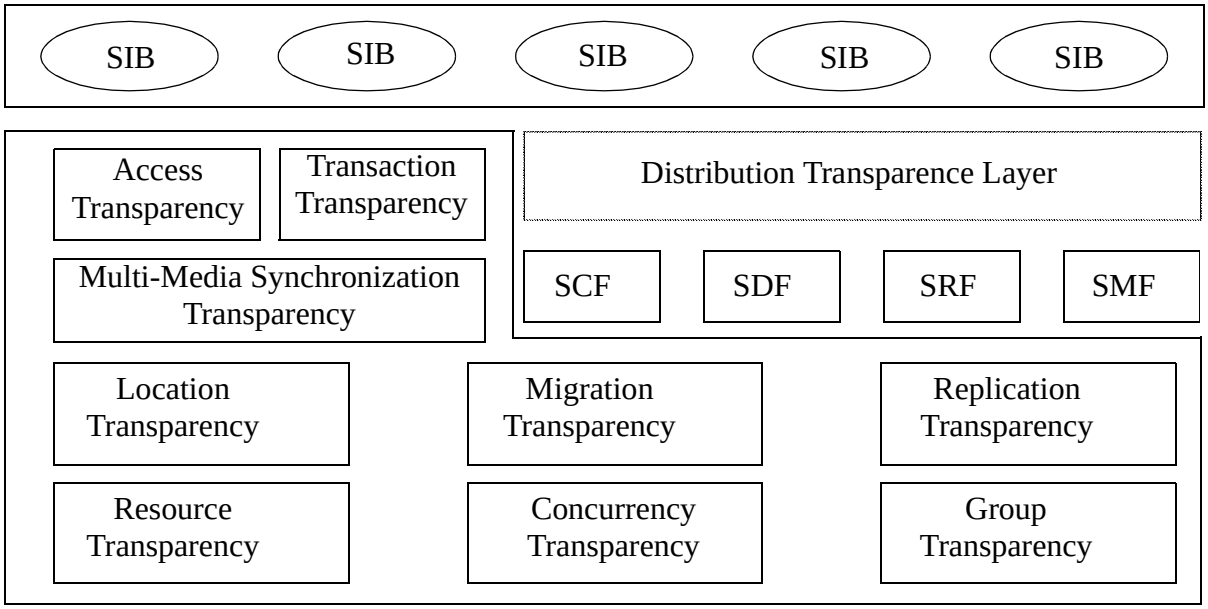
2. Migration Transparency: PCS/UPT Applications and UMTS Platforms, etc.

3. Transaction Transparency: Services for Banking Applications.

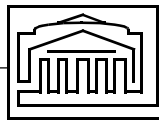
4. Replication Transparency:
   Mobile User's Profile (e.g. UPT User Profile) may be replicated on multiple nodes (SDF) in the originating/terminating/ home networks. SCF in the originating network can obtain this data in a replication transparent way.

5. Group Transparency:
   - Emerging Telecommunication Services increasingly demand various forms of user participation requiring different topological forms of connection.

   - General architectural solution for "Distributed Cooperative Applications", "Conferencing Applications", etc.



ODP Distribution Transparencies in IN-Distributed Platform



5. CONCLUSION

- Telecommunications constitute a significant application domain of ODP. ODP Provides:

  - Distributed Object Model.
  - Modelling Frameworks (viewpoints).
  - Management Framework
  - Interoperability Model,
  - Distribution Transparency Model.

- Increasing convergence taking place between architectures in telephony and distributed computing (such as TINA).

  - ODP concepts are inherited by Telecommunication Architectures; conversely these architectures bring specific issues for consideration by ODP.

- Redefinition of IN Model in the Object-Oriented Paradigm, guided by ODP Modelling Techniques.

1. IN-Service: An object-oriented software system distributed across many parts of the network.

2. IN-Platform: Finally converge into a Policy-Driven Configurable and Customisable Object-Based "Distributed Processing Platform".

- Long-Term IN Architecture = IN + ODP + OSI + TMN + B-ISDN...
  =IN (CS-n).
  The smaller the "n" the better it is!