On the Notions of Abstraction, Consistency, and Design in the ODP Framework of Viewpoints

Kazi Farooqui, Luigi Logrippo,
Department of Computer Science,
University of Ottawa,
Ottawa K1N 6N5, Canada.
(Internet: farooqui|luigi@csi.uottawa.ca)

Abstract

One of the most fundamental systems analysis and design principle is that of "abstraction". Essentially, the purpose of abstraction is to clarify or highlight some features of a problem by concealing others. The set of viewpoints identified in the ODP architecture is merely a pragmatic classification of concerns. A viewpoint leads to a representation of the system with emphasis on a specific set of concerns, and the resulting representation is an abstraction of the system, i.e., a description which recognizes some distinctions that are relevant to the concern and ignores others. The viewpoint models exhibit very subtle concepts with respect to the notion of abstraction and consistency between them. They offer a very powerful structuring paradigm suitable for a design activity. In this paper we explore the notion of abstraction, consistency between viewpoint models, and the role of viewpoint models in the design process framework. The correct understanding of the relationship between viewpoint models and their role in design activity is crucial for the construction of ODP development tools.

Keywords: Viewpoint modelling, enterprise model, information model, computational model, engineering model, technology model, viewpoint consistency, design process framework.

1.0 Viewpoint Model: System Abstraction

Different abstract models of a (distributed) system are appropriate from different viewpoints. In order to manage the complexity involved, it is necessary to consider a system from different viewpoints, rather than attempt to capture the whole picture at once [1,2]. Each viewpoint focuses on different concerns (or aspects) of the system; the whole picture is the summation of different viewpoints.

The ODP viewpoints should not be seen as architectural layers, but rather as different abstractions of the same system, and should all be used to completely analyze the system. With this approach, consistent and complete system models may be described and developed based on concepts and methods still to be designed for individual viewpoints.

Specifying a distributed system in each of the viewpoints allows an otherwise large and complex specification of a (distributed) system to be separated into manageable pieces, each focussed on the issues relevant to different members of the design team.

2.0 Abstraction Principle in the ODP Framework of Viewpoints

A viewpoint is not necessarily an abstraction of another and there is no hierarchy (or layering) between viewpoints. However, each viewpoint in some way or the other acknowledges the concerns expressed in other viewpoints but with a lesser emphasis (or detail) [3]. For example, the interaction between application components can be discussed in *several* viewpoints. In a particular viewpoint, the detail of interactions between objects is simultaneously increased and decreased. There is a more detailed description of the interaction in the terminology selected for that viewpoint. There will be a less detailed description of interaction in the terminology selected for all other viewpoints. In terms of type systems [4,5], this detail is reduced to a single name for each viewpoint. This *name* is taken to stand for the attributes of objects in a viewpoint other than the one currently considered [6].

In the computational viewpoint, for example, it is not possible to elaborate on the names that stand for attributes of objects that are discussed in other viewpoints [6]. For instance, the attributes of engineering objects (which enable, regulate, and hide distribution) are treated in the computational model as *abstractions* of the distribution (interaction) requirement between application components.

Each viewpoint-model is self-contained and complete. The difference between the models is not how much of the system they describe, but rather what aspects of the system they emphasize.

3.0 Viewpoint Languages

The Reference Model of ODP [7,8,9,10] defines a set of five viewpoint models and then identifies key generic functions which are related to these models. A set of *concepts*, *structures*, and *rules*, is given for each of the viewpoints, providing a “language” for the specification of the system in that viewpoint [9].

The specification of a (distributed) system in any viewpoint is based on the concepts of that viewpoint and satisfies the structuring rules specified for the viewpoint. For example, the *computational language* is based on the concepts of activity, operations, environment constraints, computational interface, computational object, etc. and contains the structuring rules such as interface binding rules, interface subtyping rules, invocation rules, activity rules, portability rules, etc. [9].

Any existing language can, in principle, be used for specification of a system from a particular viewpoint provided that those specifications can be interpreted in terms of relevant viewpoint concepts. What is required is to map the viewpoint-specific concepts and rules onto the formal syntax and semantics of the language.

Since ODP viewpoints are used to model different aspects of a distributed system, the language requirements for modelling the concerns in different viewpoints vary considerably from one viewpoint to another.

It is desirable to use a single formal language for the specification of all the viewpoint models. This would facilitate transformations and consistency checks between viewpoints to be developed within the single framework of the semantic model of the language. However, it is interesting to note that different modelling techniques are required to represent the specific system view in each of the viewpoints. This means that the whole system is described by the *integrated view* of the different viewpoints and therefore, by different models.

This approach immediately points to another important aspect that needs further investigation and studies: the definition of a methodology that allows to correlate the objects defined by means of different techniques in different viewpoints in order to make consistent the overall system model.

4.0 Consistency between Viewpoints

As mentioned before, a single model of a (distributed) system would be overwhelming in complexity. Therefore, we need a way to separate concerns such that we can check consistency between alternate specifications of the same system. Hence, the *viewpoint* concept - each viewpoint looks at the whole system, but uses modelling concepts specific to a defined subset of modelling concerns.

Viewpoint specifications correspond to alternate views of the same system. Since each viewpoint encompasses the whole system, we can assert consistency (between viewpoints) by identifying matching terms (concepts) in the corresponding *viewpoint specifications*.

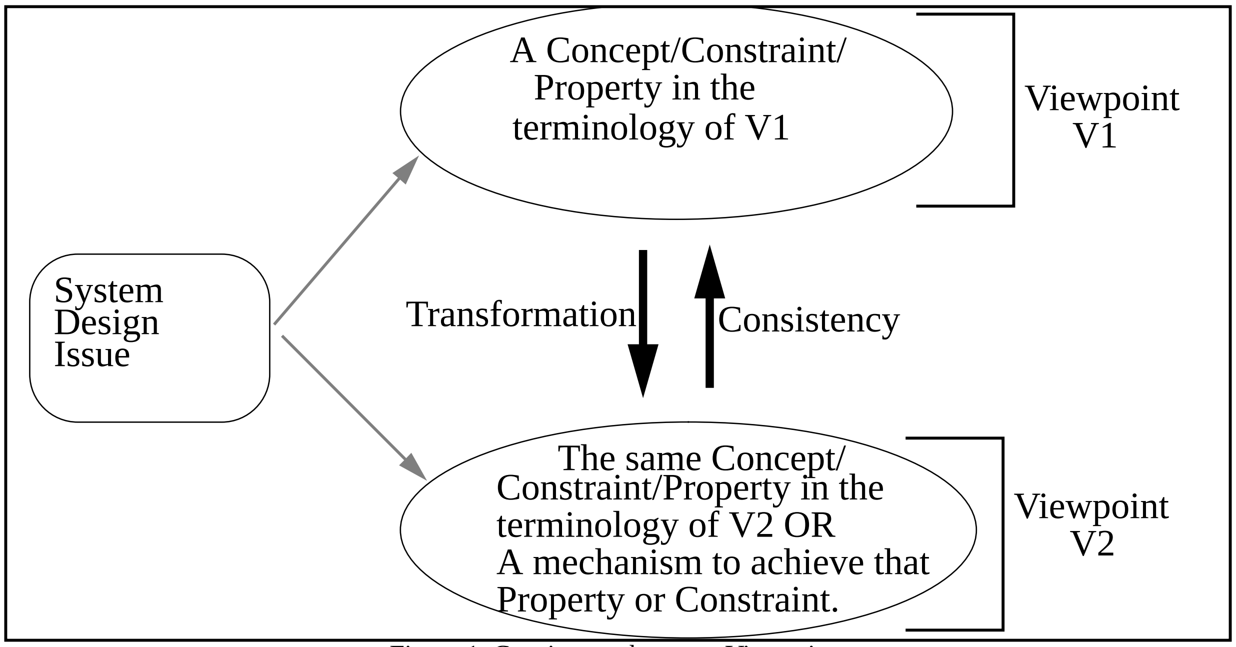
A Concept/Constraint/ Property in the terminology of V1
Viewpoint V1
System Design Issue
Transformation
Consistency
The same Concept/ Constraint/Property in the terminology of V2 OR A mechanism to achieve that Property or Constraint.
Viewpoint V2

Figure 1. Consistency between Viewpoints

A concept in a viewpoint may be represented by multiple concepts in another viewpoint; similarly multiple concepts in a viewpoint may be represented by a single (or fewer) concept(s) in another viewpoint. Additionally, concepts in one viewpoint (specification) may be abstractions or refinements of concepts in another.

Any complete specification of a system should include not only the viewpoint specifications but also the mapping between them.

Although different viewpoints address different concerns, there is a common ground between them. The framework of viewpoints must treat this common ground consistently, in order to relate viewpoint models and to make it possible to assert correspondence between the representations of the same system in different viewpoints. As mentioned in [11], achieving consistency may not be an easy job. Changes to requirements in one of the viewpoint models may require adapting requirements in other models. Moreover, the required adaptations may be quite complex; it is desired to define transformations such that minimal changes to other viewpoint models are necessitated as a result of a change in one of the models.

It may be noted that the viewpoint models are both a source of requirements and constraints on the design of a (distributed) system, and any changes to one of these models affects the specifications in other models. Although it may be argued that enterprise and information viewpoints are a source of design requirements, and computational, engineering, and technology viewpoints are a source of constraints on the system design, the design requirements and constraints originate from across all viewpoint models (albeit to different degree). The issue is identifying the requirements and constraints in all viewpoints and satisfying them in the system design. The design template may be seen as a unification of (and consistent with) all the viewpoint models.

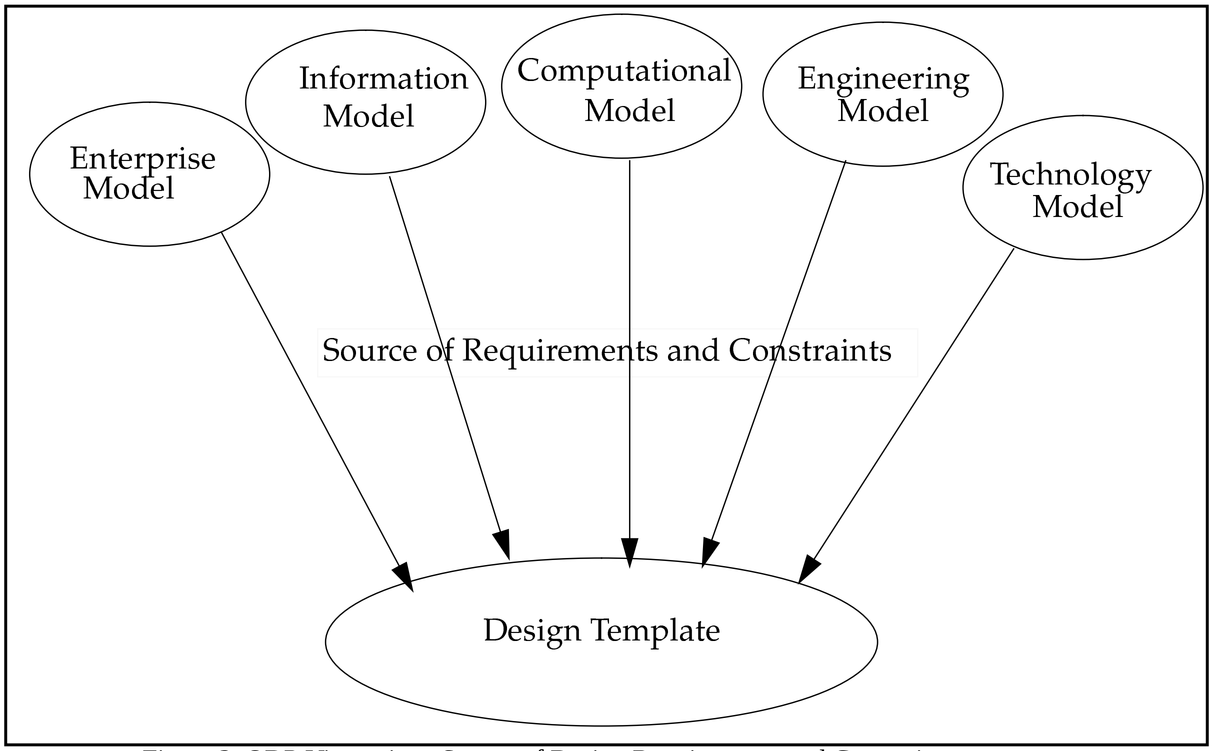

Figure 2. ODP Viewpoints: Source of Design Requirements and Constraints.

As mentioned in [12], the viewpoint models have a direct impact on the design. The more the prescription in the enterprise and information specification, the less freedom of interpretation of design requirements. Similarly, the more the prescription in computational, engineering, and technology specification, the less freedom the design has in its choice of components and their configuration.

Specifications from different viewpoints may be checked for consistency (by applying some transformations) in order to ensure that incompatible or otherwise contradictory requirements are not placed in individual viewpoint models.

Formally, one can identify the relationship or consistency constraints that must be satisfied to demonstrate that a specification of a (distributed) system in one viewpoint language is consistent with the specification of the same (distributed) system in other viewpoint languages. The consistency rules specify valid transformations of a specification in one viewpoint to a specification in another.

The consistency constraints ensure that the specification of the system in different viewpoints are not in conflict with respect to the structuring rules of the corresponding viewpoints. However, consistency constraints are not sufficient to ensure exact equivalence of the specifications. As stated in [9], equivalence is not decidable, in general, since to do so requires the validation of assertions relating to the meaning of concepts (terms) in each specification.

5.0 Viewpoint Modelling

This section presents a brief and informal review of what is involved in *viewpoint modelling* before dwelling into the issue of their use in *system design*. It is apparent that viewpoints can be thought of as constituencies of concerns involved in the system specification process. The system specification concerns that are addressed in the individual viewpoints are outlined below. It is not intended to give a 'formal' treatment to the concepts that arise in a viewpoint. Instead individual concerns specific to a viewpoint are itemized[1] in order to relate them in inter-viewpoint consistency exercise performed in section 7.

5.1 Enterprise Modelling: The *enterprise modelling* deals with the objectives of the system. The main concerns addressed in the enterprise modelling are:
(Purpose + Scope + Role + Policies + Obligation) of SYSTEM

Enterprise Modelling allows us to make statements such as:
ER1: What is the *purpose* of the system (or its components) in the enterprise.
ER2: What is the *scope* of the system (or its components) in the enterprise.
ER3: What is the *role* of the system (or its components) in the enterprise.
ER4: What *policies* are associated to the system (or its components) in the enterprise.
ER5: What are the *obligations* of the system (or its components) in the enterprise.
ER6: What are the *distribution* requirements of the system (or its components).
ER7: What are the *interactions* between the system and its environment.
ER8: What are the application-specific *requirements* of the enterprise from the system.

The *enterprise specification* is composed of a combination of these kinds of statements. These statements are made with respect to the system or its components, called *enterprise objects*. The specification from this viewpoint captures requirements that justify and orient (impact) the design of the system. The enterprise specification is at the most abstract level of detail suitable for representing user concerns and requirements.

5.2 Information Modelling: The *information modelling* deals with aspects related to information content of the enterprise. The concerns addressed in the information viewpoint are:
Identification of (Information Objects + Quality Attributes of Information Objects + Manipulations on Information Objects + Rules/constraints for Information manipulation + Relationship between Information Objects + Semantics of information stored and exchanged between components + Information Flows) of SYSTEM.

Information objects are information elements or structures of information. Information objects define the subset of the information content of the enterprise.

The specification of the system from the information viewpoint consists of the following statements:
IN1: What are the *information objects* of the system.
IN2: What are the *quality attributes* of the information objects of the system.
IN3: What *manipulations/processing* can be performed on the information objects of the system.
IN4: What are the *rules* and *constraints* for information manipulation
IN5: What is the *relationship* between information objects.
IN6: What *semantics* a human would associate with the information stored in and exchanged between information

1. Although, some of the items overlap w.r.t. their scope, the intention is to relate them with the concerns in other viewpoint(s).

objects.
IN7: What are the *sources*, *sinks*, and *information flows* in the system.

5.3 Computational Modelling: The *computational modelling* deals with the functional decomposition of the system into components, called *computational objects*, which are candidates for distribution and identification of interactions between these components. It consists of:

Identification of (Computational Objects + Activities that occur within Computational Objects + Interfaces of Computational Objects + Operations of Computational Interfaces + Behavior observable at Computational Interface + Role of Computational Interfaces + Environment Constraints associated with Computational Objects and their Interfaces + Interactions between Computational Interfaces) of SYSTEM.

The *computational modelling* activity consists of the following concerns:
CO1: What are the *computational objects* of the system.
CO2: What *activities* occur within the computational objects of the system.
CO3: What are the *interfaces* of computational objects of the system.
CO4: What *operations* can be invoked to/from the computational interfaces.
CO5: What *behavior* is observable at the computational interfaces.
CO6: What is the *role* of the computational interface.
CO7: What *environment constraints* are associated with the computational objects and their interfaces.
CO8: What *interactions* are possible between computational objects (interfaces).

5.4 Engineering Modelling: The *engineering modelling* deals with the infrastructure required to support the system components and interaction between them. It is concerned with:

Identification of (Infrastructure Objects and their configuration required to support the distribution of components) of SYSTEM.
Infrastructure objects are *engineering objects* which are either obtained from computational objects or provide specific distribution support functionality.

5.5 Technology Modelling: The *technology modelling* is concerned with the identification of technology artifacts to support the system or its components.

6.0 Design Methodology based on ODP Viewpoints

As mentioned in [13], there are a number of ways to interpret the concept of *viewpoints*. The basic interpretation of a viewpoint is that of a *constituency framework* - expression of concerns of different players involved in system development. This section and the following one presents two alternatives of using viewpoints in a *design process framework*.

Although there is no (explicit) ordering or hierarchy between viewpoints, the issue in the *design process framework* relates to the *consistency* between viewpoints. The specification in a given viewpoint must reflect the requirements posed in other viewpoints and not contradict them.

The *design process framework* is used to illustrate the relationships that can be identified between viewpoints, and is not intended to suggest that other relationships are unsuitable. The design process interpretation ascribes to each viewpoint a different role in the design process.

Although, as suggested in section 8, the viewpoints can be used at all stages of the design of the system in order to ensure consistency between viewpoint specification during the design process, some viewpoints play a predominant role (than other viewpoints) in particular phases of system design. In particular some stages of system design trajectory are heavily dependent on the specifications in some viewpoint(s) than others. Of particular importance is the observation that some viewpoint specification(s) capture much details of a certain stage of design trajectory than others.

In particular, the system specification in one viewpoint is based upon requirements expressed in some other viewpoint(s). This relationship between viewpoints, which is related to the concerns in different phases of system design, is illustrated in figure 3.

The system design process potentially involves all the viewpoint specifications. One specification may be responsible for the generation of other specifications through the application of appropriate transformations.

The viewpoints can be used to structure the specification of a (distributed) system, and can be related to a design methodology. As outlined in [7], design of the system can be regarded as a process that may be subdivided into phases related to different viewpoints. These phases and their relationship to the ODP viewpoints are shown in figure 3.

In the figure, the four major phases in the system design trajectory are considered: *requirement capture* (*and analysis*), *functional specification*, (*detail*) *design*, *implementation*. This corresponds to the three major design steps identified in the classical waterfall model described in [14] and [15]. The phases related to testing and maintenance are not shown. Each of the viewpoints can be used as problem analysis technique as well as a solution space of the relevant issues of the problem domain.

Apart from their use in the (parallel and) alternate system specification, the use of the viewpoints can be organized to assist the design trajectory from requirements specification to final design and implementation.

1. *Requirement capture and analysis*: The classical "waterfall" model of the software life cycle [14], [15], begins with *requirement analysis and definition* phase. According to the model, the system's services, constraints and goals are established, in this phase, by consultation with system users. This is precisely the concern of the enterprise viewpoint in the ODP framework of viewpoints.

   The enterprise view covers the enterprise objectives of an information system. It focuses on the *requirements*, *objectives*, and the *role* of the system within the organization. It is the most abstract of the ODP framework of viewpoints stating high-level enterprise requirements. This allows the designer to develop a closed (i.e., bounded) model which represents all the real world requirements which the designer must incorporate, later in the design trajectory, into the final realization of the system.

The classical "waterfall" model identifies a single phase corresponding to "system design". For complex (distributed) systems, it is not possible to achieve an implementable design in a single step from requirement specification. The ODP framework of viewpoints allows the identification (and specification) of an intermediate design step-the *architectural* or *functional* design of the system which is followed by a detailed design of the system.

2. *Functional specification (architectural design)*: This step consists of decomposition of system into *functional* or *architectural* components and identification of interconnections between them using the requirement definition as the base. The interaction requirements of the functional components are identified. This is the domain of the computational viewpoint. The computational specification of the system contains detailed design constraints (reflected in terms of environment constraints associated with computational objects, computational interfaces, and interactions between them). This forms the basis for the detailed system design in the next step.

   Apart from *functional decomposition,* it is also possible to specify the system based on its information content [16,17,18]. This activity, referred to as *information modelling* is concerned with the semantics of information, information processing activities, and information flow in the system. The *conceptual decomposition* of the system is performed as part of information modelling.

   Although the information model does not directly impact the system design process, the computational viewpoint plays a central role in the design process. While the computational viewpoint helps in the *functional* decomposition of the system, the information viewpoint enables the *conceptual* decomposition of the system. Together they help in constructing the architectural specification of the system. For example, the information model ascribes meaning to the information that is exchanged in interactions between *functional* components identified in the computational model.

3. (*Detail*) *design*: The detailed design step fills in the gap of the architectural design by completing the design step corresponding to some special system requirements such as *distribution* of the system. The *functional* or *architectural* components identified in the previous design step need to interact in order to perform the objectives of the system identified in the enterprise viewpoint. The components required to support the interactions between *architectural* components and the configuration of the supporting components are identified in the engineering viewpoint. The functionality of the supporting components is consistent with the requirements of interaction (environment constraints) between the *architectural* components identified in the computational view. Moreover the identified *architectural* components may be decomposed, in the final design step, to enable mapping onto an implementable design.

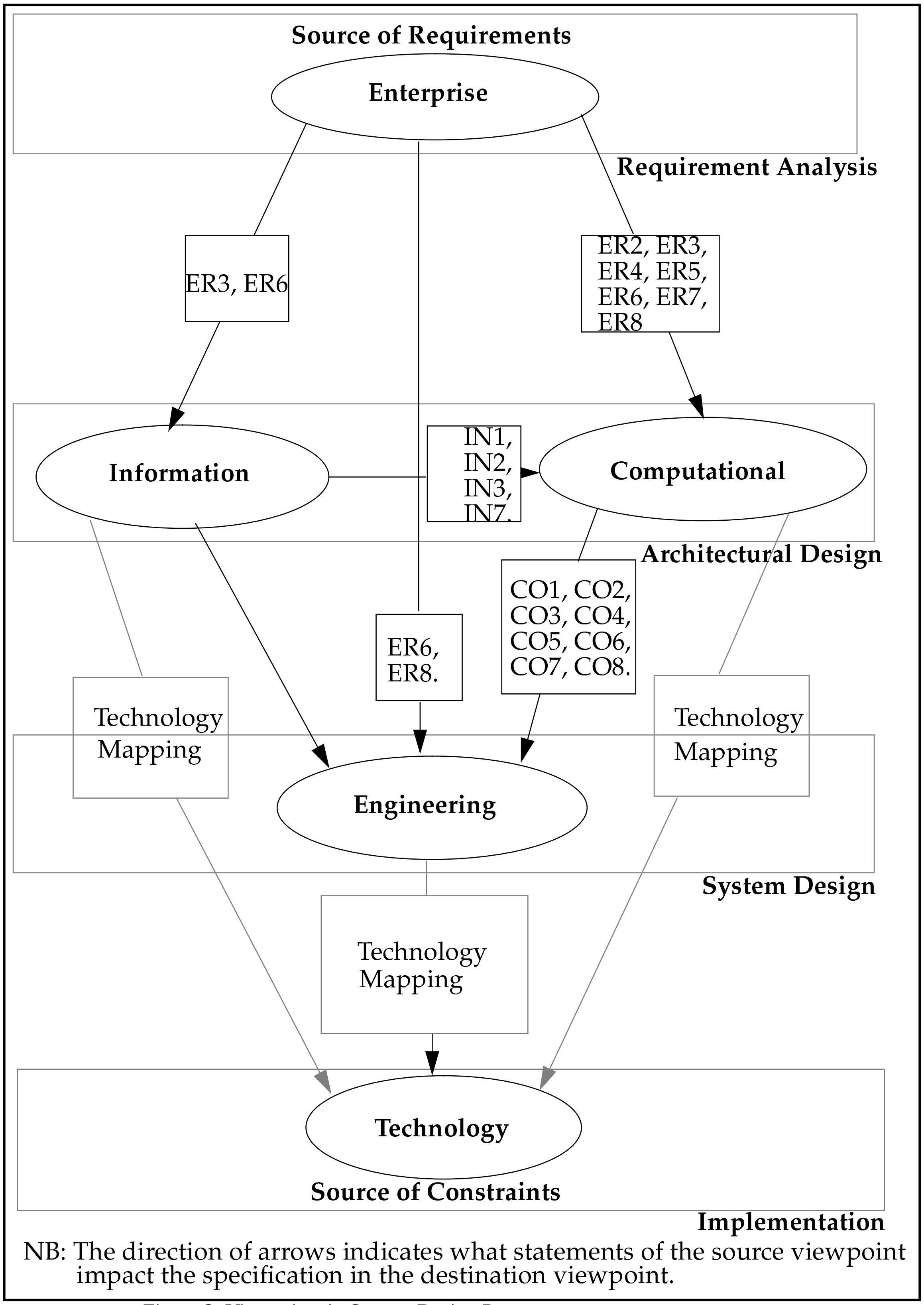

Figure 3. Viewpoints in System Design Process

4. *Implementation*: The implementation phase corresponds to giving technology mappings to the system design obtained in the previous step. The identified system components (functional components and supporting components) are mapped onto the available technology artifacts. This step relates to software coding using programming languages, embedding into computer systems, and identification of communication protocols (networks) for connecting system components.

However, from an application designers perspective, the issues related to enterprise and information viewpoint are only visible in terms of application requirements they generate; issues related to related to engineering and technology viewpoint are invisible, buried somewhere in the distribution support environment being used. As shown in figure 4, the computational model plays an important role in bridging the gap between requirements and solutions.

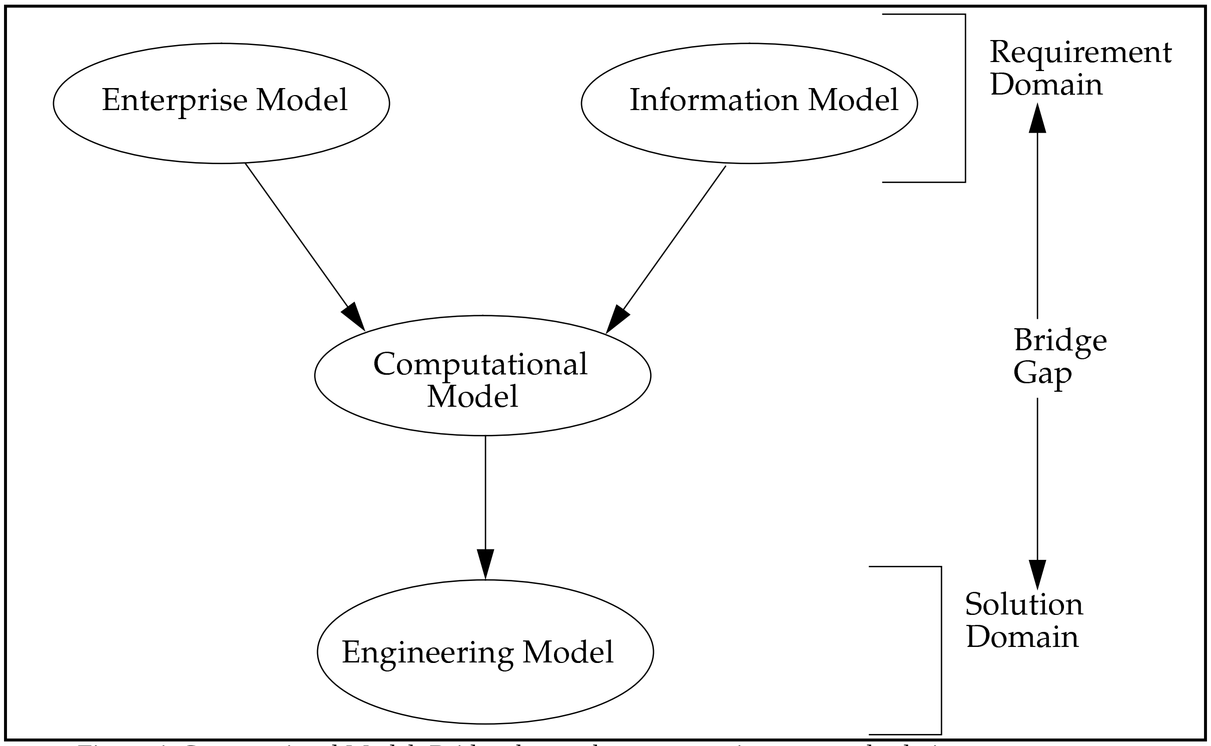

Figure 4. Computational Model: Bridge the gap between requirements and solution

While the enterprise viewpoint is the ultimate source of (design) requirements, the technology viewpoint is the final source of (design) constraints (what can be implemented with the available technical artifacts). The enterprise and technology viewpoints together delimit the design space for the designer of a (distributed) system.

However, it should be noted that the specification of the system in one viewpoint should be consistent with the system specification in all other viewpoints during all phases of design trajectory.

7.0 Viewpoint Consistency in the Design Process Framework

In the following we identify the consistency relations between viewpoint models and show how the statements made in a viewpoint specification impact the system specification in another viewpoint, as illustrated in figure 3. This mapping process is not meant to give a 'formal' interpretation but to informally explain the relationship between each of these models.

Enterprise and Computational Specifications: The enterprise specification is the source of design requirements. Hence it has a direct impact on other viewpoint models.
With respect to the enterprise concerns identified in section 5.1, the following enterprise concerns affect the architectural design of the system (specified in the computational viewpoint):
ER2: The *scope* of the system (or its components) in the enterprise helps in the identification and subsequent delimit-

ing of the functionality of computational objects.

ER3: The *role* of the system (or its components) in the enterprise decides what activities will be performed by computational objects. These roles may be reflected at computational interfaces. This mapping need not be one-to-one, for example, some roles (of enterprise objects) may be distributed amongst computational objects.

ER4: The *permissions* and *prohibitions* of the system (or its components) in the enterprise has an impact on the constraints on the activities performed by computational objects and on the interactions between them.

ER5: The *obligations* of the system (or its components) in the enterprise are to perform some enterprise *function*s; this decides the processing (or behavior) of the computational objects.

ER6: The *distribution* requirements of the system (or its components) decides the interfaces of computational objects.

ER7: The *interaction*s between the system and its environment identify the computational objects that constitute the man machine interface.

ER8: The (special) *requirements* of the enterprise from the system have an influence on what *environment constraints* (such as security, distribution transparency, communication QoS, etc.) are associated with computational objects or their interfaces.

Enterprise and Information Specification: The enterprise requirements also impacts the information modelling of the system. Apart from establishing the "ownership" of information, i.e., which enterprise roles are associated with managing which information, there is also the issue of associating information with enterprise (business) functions. The latter has an impact on the design process, for example having identified the enterprise functions it is possible to define the information associated with it. The resulting information specification will get refined as it is decided exactly what information objects the computational specification will deal with.

The information viewpoint must capture information about the system sufficient to allow the roles, policies, etc., of the enterprise model to be performed, for example, how various enterprise roles are held and by which objects.

There are few rules that apply to the consistency between enterprise and information viewpoint because there are few prescriptive rules in these languages (compared to computational and engineering viewpoints).

ER3: The *role* of the system (or its components) in the enterprise decides what manipulations can be performed on the information objects.

ER6: The *distribution* requirements of the system (or its components) decide the source, sink, and information flows, in the information model of the system.

Information and Computational Specification: The information model defines the overall state for the part of the system being modelled. The aspects of the computational specification which affect the information objects should be consistent with the information specification.

The computational viewpoint describes how the information described in the information viewpoint is processed in the computational objects and which information, modelled, in the information viewpoint, is exchanged between computational interfaces (for example, as arguments to operation invocations).

Similarly, the computational type system [9] requires that operation and termination names be known to the recipients. However, it does not guarantee that the recipients agree on the semantics of the names. The agreement on the semantics of the exchanged information is performed in the information model.

IN1: The *information objects* of the system are processed (or manipulated) by computational objects. There is not necessarily a one to one correspondence between information and computational objects. Moreover, the information objects may correspond to computational objects that perform information processing or to the computational objects that store information.

IN2: The *quality attributes* of information objects may require additional environment constraints to be associated with computational objects or their interfaces.

IN3: The *manipulations/processing* that can be performed on information objects affects the exact nature of computational operations that are possible within the computational objects. This should be consistent with enterprise requirement ER3.

IN7: The sources and sinks of *information flows* helps identify client and server roles of computational objects and their interfaces.

Computational and Engineering Specification: The computational specification has a direct impact on engineering specification. The engineering model of the system animates its computational model. A detailed discussion of the relationship between computational and engineering models is given in [19].

CO1: The *computational objects* can be mapped directly, or refined into (a number of) basic engineering objects.
CO2: The *activities* of the BEOs are the one's of the corresponding computational objects.
CO3: The *computational interfaces* may map onto single BEO or onto an interface of a BEO. The computational interfaces cannot be decomposed.
CO4: The *operations* invoked to or from BEO interfaces correspond to computational operations.
CO5: The computational *behavior* must be preserved in transformation to engineering specification.
CO6: Since computational interfaces cannot be decomposed, there is an interface to BEOs corresponding to computational interface with the identical *role*.
CO7: The computational interactions are performed in a distribution transparent manner. The *environment constraints* associated with computational object and their interfaces are the main source of requirements that decide the composition of structures of supporting objects within the channel between BEOs.
CO8: All *computational interactions* must be preserved as interactions between interfaces of BEOs.

Enterprise and Engineering Specification: Although, in general, it must be ensured that engineering specification is consistent with the enterprise specification, there are certain enterprise concerns which have a direct impact on the engineering design. They are:
ER6: The *distribution requirements* of the system have a direct bearing on the location of basic engineering objects.
ER8: The (special) *requirements* of the enterprise decide the distribution transparency, communication, and other supporting objects within the channel between BEOs.

Information and Engineering Specification: The information viewpoint defines the types of information exchanged and therefore determines the required communication service for exchanging information objects. Moreover, policies concerning distribution transparency are specified in the information viewpoint.

Information, Computational, Engineering to Technology Specification: The mappings to technology artifacts are identified for information objects, computational objects, and engineering objects. The technology mappings from information, computational, and engineering specifications include:
1. Information objects can be realized as databases (or parts thereof).
2. Computational objects (application components) can be realized as technology objects such processes, computer programs, etc.
3. Engineering objects (protocol objects, transparency objects) can be mapped on to technology objects such as protocol software modules, transparency software modules, etc.

8.0 Recursive use of Viewpoints in System Design

This section illustrates how viewpoints can be used *recursively* in the design of (distributed) system and of its components.

The viewpoint models can be used at different levels of granularity. They can be used on the large scale to specify complete systems, or on the small scale to talk about system components. As discussed below, the process is recursive.

The methodology is based on the recursive use of viewpoints for the development of the specification of (distributed) systems in different stages of design trajectory. In this methodology the ODP concept of viewpoint is treated as a "design or analysis tool" that aids in the identification and refinement of system into its constituent components. The starting point are the enterprise policy statements relevant to the (distributed) system, and then the other viewpoints are used to determine the constituent (or supporting) objects of the system. The same procedure is then applied recursively on the constituent (or supporting) objects until the specified objects are detailed enough to be mapped to the real resources (i.e., a mapping is found to technology artifacts in the technology viewpoint).

Similarly, objects identified in a given viewpoint specification can be specified from different viewpoints. For example an *engineering object*, say a *transparency object*, can be specified using the enterprise viewpoint (its objectives), information viewpoint (information requirements for that part of the system), computational viewpoint (the computational activities, interfaces, and interactions associated with that object), engineering viewpoint (supporting objects, if any, required for complete realization of that object), and technology viewpoint (what technical artifacts can be used for the realization of that object).

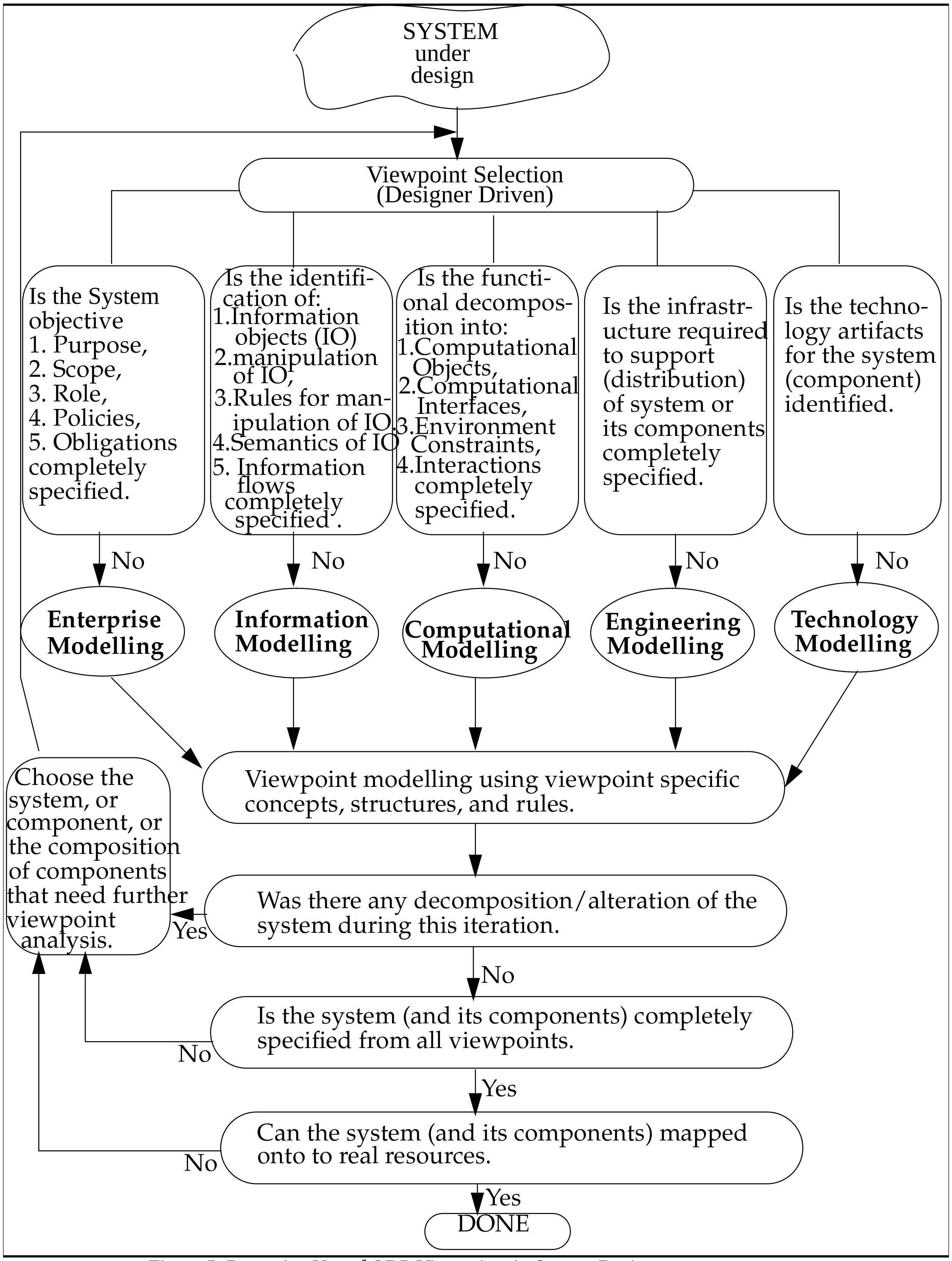
SYSTEM under design
Viewpoint Selection (Designer Driven)
Is the System objective 1. Purpose, 2. Scope, 3. Role, 4. Policies, 5. Obligations completely specified.
Is the identification of: 1.Information objects (IO) 2.manipulation of IO, 3.Rules for manipulation of IO, 4.Semantics of IO 5. Information flows completely specified.
Is the functional decomposition into: 1.Computational Objects, 2.Computational Interfaces, 3.Environment Constraints, 4.Interactions completely specified.
Is the infrastructure required to support (distribution) of system or its components completely specified.
Is the technology artifacts for the system (component) identified.
No
Enterprise Modelling
Information Modelling
Computational Modelling
Engineering Modelling
Technology Modelling
Viewpoint modelling using viewpoint specific concepts, structures, and rules.
Choose the system, or component, or the composition of components that need further viewpoint analysis.
Was there any decomposition/alteration of the system during this iteration.
Yes
Is the system (and its components) completely specified from all viewpoints.
Can the system (and its components) mapped onto to real resources.
DONE

Figure 5. Recursive Use of ODP Viewpoints in System Design

1. *Recursive use of enterprise viewpoint*: is used to identify the enterprise requirements of the system or of its components.
2. *Recursive use of information viewpoint*: is used to identify the information (processing) requirements associated with the system or with its components. Additional information objects (and hence the manipulations, quality requirements, information flows, etc.) associated with the system (or its components) may be identified. This activity is dependent on the designer and on the available technology mappings for the identified *information concepts* (information objects, etc.).

3. *Recursive use of computational viewpoint*: is used to identify the (additional) computational objects (and hence the computational interfaces, interactions between computational interfaces, environment constraints associated with computational interfaces) into which the system (or its components) can be refined. This activity is dependent on the designer and on the available technology mappings for the identified *computational concepts* (computational objects, etc.).
4. *Recursive use of engineering viewpoint*: is used identify the required (and possibly additional) supporting objects for the engineering of the system (or its components). This activity is dependent on the designer and on the available technology mappings for the identified *engineering concepts* (engineering objects, etc.).
5. *Recursive use of technology viewpoint*: is used to identify the technology artifacts for the realization of the system or its components.

Different viewpoints allow specifiers to view distributed systems from different abstractions, allowing the complex problem of the specification of distributed systems to be divided into smaller, more manageable parts.

9.0 Conclusion

This paper has explored the subtle concepts of abstraction and consistency between the viewpoint models and analyzed them in the perspective of a design process framework.

We have identified different approaches for using viewpoint models in a design activity. We have proposed methodical approaches to support the design trajectory of (distributed) systems within the architectural framework of ODP viewpoints.

What is crucial in all these approaches is to maintain the consistency between viewpoint specification during the design process and to ensure that the design reflects the requirements and constraints specified in the viewpoint models.

Any future ODP design tool must take into account these aspects related to abstraction, consistency, and the applicability of viewpoints models in the design process framework.

Acknowledgment: We would like to express our sincere thanks to Dr.Andrew Herbert, APM Ltd., Cambridge, U.K., Mr.Jean-Bernard Stefani, CNET, France, Dr.Kerry Raymond, University of Queensland, Australia, and to Jonathan-Legh Smith, British Telecom, U.K. for the numerous discussions we have had with them on the issues related to the topic.

10.0 References

1. P.F. Linington. Introduction to Open Distributed Processing Basic Reference Model, Proceedings of the IFIP TC6 International Workshop on Open Distributed Processing (eds: J.d. Meer, V. Heymer, R. Roth), Berlin, Germany, September 1991 (North Holland).
2. K. Raymond. Reference Model of Open Distributed Processing: A Tutorial, Proceedings of the IFIP TC6 International Conference on Open Distributed Processing (eds: J.d. Meer, B. Mahr, O.Spaniol), Berlin, Germany, September 1993 (North Holland).
3. ANSA: An Engineer's Introduction to the Architecture, TR.03.02, Advanced Projects Management Limited, Cambridge, U.K., November 1989.
4. A.J. Watson. Types and Projections, APM/RC.258.03, Advanced Projects Management Limited, Cambridge, U.K., April 1992.
5. J.Indulska, K. Raymond, M. Bearman. A Type Management System for ODP Trader, Proceedings of the IFIP TC6 International Conference on Open Distributed Processing (eds: J.d. Meer, B. Mahr, O.Spaniol), Berlin, Germany, September 1993 (North Holland).

6. ANSA Reference Manual, Volume A., Release 01.01, Advanced Projects Management Limited, Cambridge, U.K., July 1989.
7. Draft Recommendation ITU-T X.901 / ISO 10746-1: Basic Reference Model of Open Distributed Processing - Part-1: Overview.
8. Draft International Standard ITU-T X.902 / ISO 10746-2: Basic Reference Model of Open Distributed Processing - Part-2: Descriptive Model.
9. Draft International Standard ITU-T X.903 / ISO 10746-3: Basic Reference Model of Open Distributed Processing - Part-3: Prescriptive Model.
10.Draft Recommendation ITU-T X.904 / ISO 10746-4: Basic Reference Model of Open Distributed Processing - Part-4: Architectural Semantics.
11.M.v. Sinderen, J. Schot. An Engineering Approach to ODP Systems Design, Proceedings of the IFIP TC6 International Workshop on Open Distributed Processing (eds: J.d. Meer, V. Heymer, R. Roth), Berlin, Germany, September 1991 (North Holland).
12.G.Cowen, J.Derrick, M.Gill, G.Girling, A.Herbert, P.Linington, D.Rayner, F.Schulz, R.Soley. PROST Report of the Study on Testing for Open Distributed Processing, Advanced Projects Management Limited, Cambridge, U.K., May 1993.
13.D.Iggulden, O.Rees, R.v.d.Linden. Architecture and Frameworks, APM.1017.00.03, Advanced Projects Management Limited, Cambridge, U.K., June 1993.
14.I.Sommerville. Software Engineering (4th Edition), Addison-Wesley Publishing Company, 1993.
15.Pressman. Software Engineering (2nd Edition).
16.J.J.v.Griethuysen. Enterprise Modelling, a Necessary Basis for Modern Information Systems, Proceedings of the IFIP TC6 International Workshop on Open Distributed Processing (eds: J.d. Meer, V. Heymer, R. Roth), Berlin, Germany, September 1991 (North Holland).
17. S.Rudkin. Modelling Information Objects in Z, Proceedings of the IFIP TC6 International Workshop on Open Distributed Processing (eds: J.d. Meer, V. Heymer, R. Roth), Berlin, Germany, September 1991 (North Holland).
18.L.Cerchio, N.Curci, O.Ghisio, R.Saracco, M.Volpe. Information Modelling: Paving the Way Towards the Telecommunication Management Network of Future, CSELT Technical Report, Vol.18, No.6, December 1990.
19.K.Farooqui. Viewpoint Transformation, Proceedings of the IFIP TC6 International Conference on Open Distributed Processing (eds: J.d. Meer, B. Mahr, O.Spaniol), Berlin, Germany, September 1993 (North Holland).