Models of Design Requirement

Ömer Akin and Ipek Özkaya
Carnegie Mellon University
School of Architecture
Pittsburgh, PA 15213
USA

ABSTRACT

Case studies show that significant proportions of design errors and failures are linked to poor requirement specification during both early stages of design and as changes occur. Computational requirements engineering as a front-end to design iterations is a promising area addressing these problems. In other design disciplines, such as in software engineering, requirement engineering has given significant product improvements. In this paper, we present a state-space representation of requirement models for architectural design. The purpose of requirement modeling in design is to create a process by which requirements can be converted into working design solutions through front-end validation. We suggest three models of requirement specification, co-evolutionary [CoM], multiple domain [MDM] and single domain [SDM] models, that can facilitate this effort. Taken together all three models provide a full set of logical permutations of requirement-solution "worlds" and "operations." We compare each model against the others in terms of facilitating change management and computability.

1 INTRODUCTION

There is a strong need for computational tools that support heterogeneity and iterative design for initial design stages. One of the key tasks of early phases of design is managing the ill or partially defined requirement sets and extracting usable information from them. Design requirements have been utilized in tools, which focus on generative design targeting the problem of design constraint management and alternative solution search. Similarly, a significant amount of research concentrates on building data modeling to solve design data and information management. However, none pursue the design processes prior to the existence of a design solution where design requirements steer the design activities. More importantly, none tackle how design requirements affect the end product as design progresses. The reason for modeling requirement specification during the early phases of design is to computerize the tasks towards "improving" both the process and the end product.

The word requirement is usually used in a general sense meaning "needs". These needs are more often than not used as emerging requirements essential in design generation. Requirement modeling is a step of inception in design, which also

aids in design iteration [Figure 1]. Design requirements are intended to serve as a basis for (1) agreement between clients and professionals on the scope of the work to be done, (2) reducing the design effort by focussing on relevant design tasks, (3) early estimates of costs and schedules, (4) validation and verification of designs, (5) managing revisions and modifications of scope, and (6) developing new requirement sets.

A requirements modeling approach is an attempt to formalize this fuzzy domain. Requirement, inconsistencies and dependencies can be formally organized with correct and situated models. Requirement dependency captures required behavioral dependency relationships between individual design component's behaviors and specifications and how such dependency relationships are realized by the component. Such dependencies capture where one component or subsystem causes a behavior in another component or subsystem and where a specific order of relationships has to be maintained. Inconsistency dependency, on the other hand, addresses the refinement of inconsistencies or conflicts that arise. This effort regards requirements as an integral part of design solution iteration, which guides design through specification, alteration, validation, and verification stages.

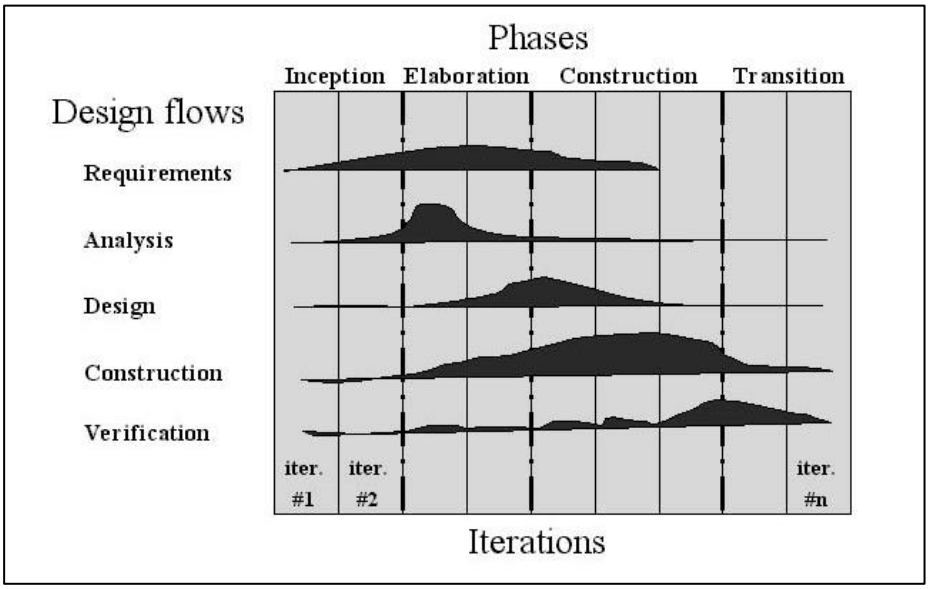

Figure 1: **The distribution of design activities through different phases (Adapted from Jacobsen et. al 1999)**

Building design research commonly cites several drawbacks in the early use of computational environments in architectural design problems, especially during the early, ill-defined, and multidimensional phases of design. Also uses of computer-aided tools in the practice have been limited to drafting tools (Akin *et al*. 1995). The users are mostly naive about the computation domain. The use of software being

limited to drafting tools, result in early discarding of innovative tools. Visual representation tools dominate the demand from the practitioners as opposed to tools that provide cognitive aid in design.

Traditionally design alternatives emerge as a result of the analysis and specification of conceptual ideas. In this research, the analysis and specification stages match the computational requirement analysis framework. So far, the initial solution investigation activities undertaken by a designer do not have a counterpart in the computational medium. These activities are potential candidates for requirements used in the digital world.

For example, designers draw bubble diagrams to understand spatial relationships. These diagrams usually become an initial starting point where most of the spatial requirements are investigated. The designer may set these aside and start plan layout investigations. This understanding usually comes from the diagramming stage. Similarly, three dimensional space explorations are another set of diagrams that designers use in problem understanding. Thus, a requirement modeling framework is a medium where the designers can explore the given data, generate implicit ones and iterate between various design stages [Figure 2].

In section two, we present related work that motivates our work. In section three, we present three models, which provide a conceptual framework towards the development of a computational requirement modeler. Before concluding and outlining the future directions of the research in section five a comparative view of these models is presented.

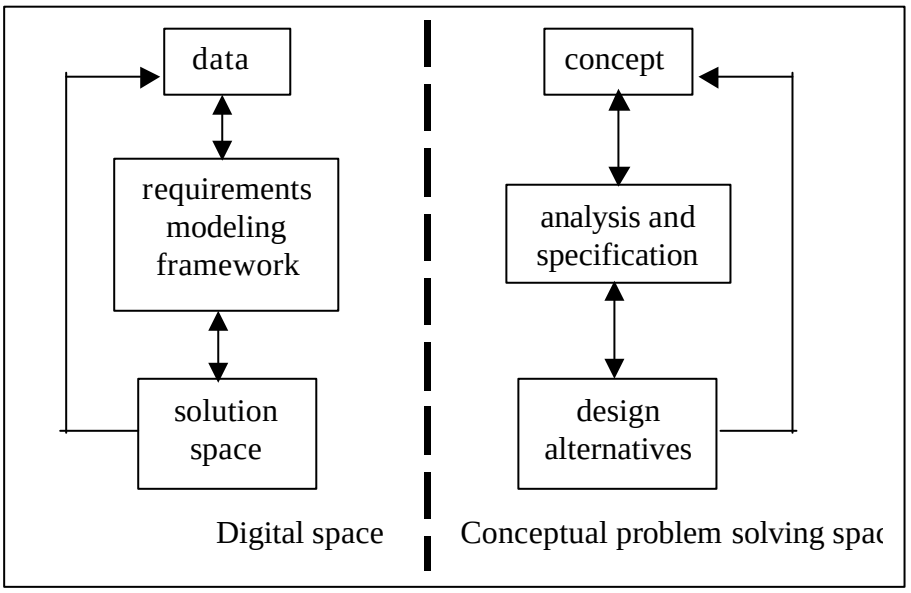

Figure 2: **The mapping between the traditional design problem solving and the computational medium. Requirements specification becomes an interface between a concept, the data and the real world object, the design solution**.

2 BACKGROUND

2.1 **Requirements in Building Design**

Requirements capture traditionally takes place during the architectural programming [AP] stage of design delivery. AP in building design has an emphasis on types of data used, integration with the design process, parameter specifications, standards and the social-contextual aspects of collaborating with clients.

Publications in the field emphasize the diversity of styles and approaches (Sanoff, 1978) to specifying design requirements. This is a critical feature, which has to influence new approaches to design requirement modeling. The different styles of AP are situated along a continuum defined by two polar extremes: performance-centered versus archetype-centered specification.

The former views AP and design in a form-neutral relationship, where the specification only regulates the functions of the solution and not its form. The designer is expected to invent new forms that respond to the functional requirements of the AP. The latter approach is function-neutral, freeing the designer, instead, to reinterpret the unspecified functions of a building type using archetypal ideas to develop a solution. While neither exists in its pure form these stylistic opposites help us understand the variety of the diverse activities, clients and goals of AP, or engineering design in general (Clayton, *et.al.*, 1999).

Earlier sources characterize AP as a process of data collection and evaluation (Sanoff, 1978). Later the prevailing view changed to one of AP being a process of analysis leading to design synthesis (Pena, 1987). This is where the decoupling of the programmer and designer are also advocated through the performance-centered approach. This trend marked the shift from theoretical descriptions to practical ones specifying methods or how-tos of AP (Hershberger, 1999). In this view, AP is a series of predestined acts of data manipulation with the explicit goals of quantifying the process, reducing information overload, making data more accessible and ultimately improving design quality through the clear definition of the clients' expectations, needs and dreams (Kumlin, 1995). The outcome of this trend has been to see AP, returning to its early roots, as a way of defining the design problem that includes the client in the frequently obtuse architectural design process (Cherry, 1999).

Representing client's views in the design process, in fact, has been an unchanging motivation for AP. Starting with Sanoff's work and culminating in rk the social aspect of AP is a key ingredient of the methods proposed. Participatory strategies include clients, users, and owners of facilities not merely as data providers but also as active, ongoing data sources even as decision makers.

2.2 **Requirements Engineering in Other Design Disciplines**

The end products of architectural processes are single, each design problem results in a unique deliverable, which can be modeled in multiply successful ways.

Similarities between software engineering, industrial design, automotive design, aerospace engineering and naval engineering practices and architectural design emerge in how requirements specification is handled. Leffingwell and Widrig (2001) state that statistically it was found that largest problems found in unsuccessful projects were in requirements specification and managing customer requirements and changes. Moreover, requirements errors were found to be the highest category of errors that contributed to the overall efficacy of the end product.

The discipline of software engineering has been using requirements management tools for over two decades. Computer-aided software engineering [CASE] tools where requirement can be specified graphically using unified modeling language [UML] or formal specification is enabled have been around for quite some time. Ironically these tools take their metaphors from the architectural design processes and the drawing sets we produce to represent our design solutions from different perspectives (Leffingwell and Widrig 2001, Jacobsen et al. 1999).

In software engineering there is a distinguished sub-discipline as requirements engineering which addresses the problem of managing initial design requirements throughout the whole life cycle of the product to increase quality while decreasing cost and time of change. Automated and computational tool support is the main argument for requirements engineering in software design. Andriole (1996) and Jackson (1995) articulate the goals of requirements modeling as analysis, definition (or specification), management and modeling. Holtzblatt (1995) distinguishes between requirements definition, gathering, elicitation, and engineering. Imelinski, *et.el.*, (1991) highlight distinctions between representation and interpretation of data, ability to ask and evaluate queries, encapsulation of non-determinism, and the notion of choice. In an early study, Liskov and Zilles (1975) develop performance attributes for requirements modeling itself: reuse of prior specifications, consistency checking, supporting iterative design cycles, identifiable sources of specification data, range of applicability, extensibility of applications, and minimization of training of users. Pidd (1996) in a theoretical paper articulates some "guiding principle
modeling, which include simplicity, parsimony, incremental construction and eliminating the dominance of the data model.

Customer perspective is another area where requirements play a crucial role for the success of the end product. Nielson (1993) argues that the purpose of the product developed is to serve some benefit to the user, thus the users must be the source of the information since developers do not use the system. However, due to the gap between users and designers yet another major requirements problem is confronted, the end product does not meet the expectations, a lot of changes have to be made, the cost of production raises and critical errors may occur.

3 MODELS OF DESIGN REQUIREMENTS

An important task in requirements modeling is requirement capture; i.e. the process of finding out what best describes what is to be built. This is often done

through formal and quasi-formal representations (Nielsen 1993). The advantage of formal specifications is that a computer can process them. A specification captures a concept if the design driven from that specification can be proven to be analytically equivalent to it. Such specifications do exist in architectural design, yet are usually overlooked during the early phases of requirements specification. This process goes beyond expecting the clients to know what they require. We suggest three rival models of requirements specification:

1. Conventional requirement and solution representations in design are generally modeled to achieve intra-state transformation through a *single* operation set mediating *predefined entities*. This model (CoM) is inspired by the *coevolutionary* design literature in Artificial Intelligence. It attempts to capture design requirement information while concentrating primarily on the generative functionalities of design.
2. Multiple-domain model (MDM) with *multiple* operation sets on *predefined entities*. This model provides a formal description of design requirements, design solutions and a full suite of operational transactions between them.
3. Single-domain model (SDM) using a *single* operation set on *dynamically defined entities*. It aims at facilitating the representation and propagation of requirement decisions in design, extraction of relevant product modeling information, and mapping of requirement decisions into and out of parallel design decision domains.

In this context, an entity is a design object and an *operation* is any type of action applied to these entities in order to generate plausible design solutions. The operations, i.e. the actions of the designer, change the problem state. In a single operation set there is only one type of operation to change the design state, whereas a multiple operation set introduces different types of operations to manipulate different design states. A design state could be an intermediate solution stage, e.g. plan layout or a requirements specification, or the minimum and maximum dimensions of office spaces.

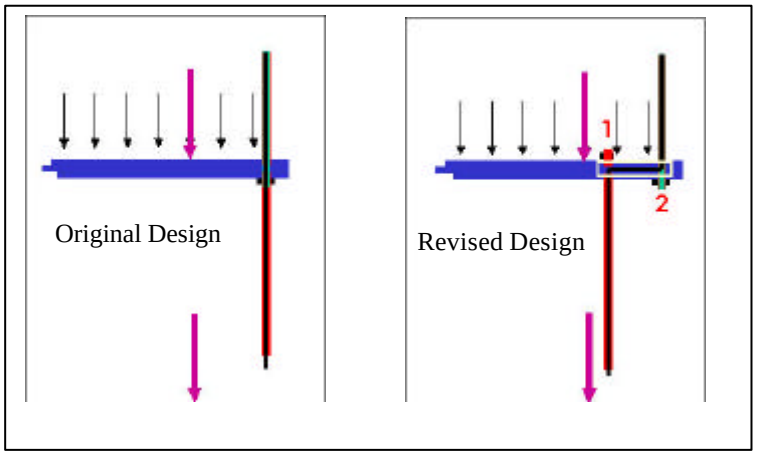

Figure 3: **Gravity load distribution of original and revised designs (Akin, 1999).**

In a dynamically defined world each operation introduces a different design step, whereas in a predefined world each operation progresses in a search space where generative solutions are sought.

In order to demonstrate the applicability and feasibility of each of the design requirement models we will use a well-known case, Hyatt Regency Hotel, Kansas City, MO as an example.

On July 17, 1981 two of the suspended walkways within the atrium area of the hotel collapsed leaving 113 people dead and 186 injured. A continuous rod was proposed to hold both second and fourth floor walkways. The rod was to carry box beams for each floor and the box beams were to carry the slab of walkways. Contractor submitted shop drawings modifying this design into two rods, one supporting the fourth floor walkway and other the second floor, from the former. The investigators concluded that the sole reason of the collapse was the substandard modification of the cable support system of the walkways, i.e. a faulty backtracking of requirements to design solution (Marshall et al. 1982). Figure 3 illustrates the design decision made on the supportive rod.

While the Kansas City Hyatt Regency Hotel collapse involves issues beyond computational design requirement management, it still provides a good example of demonstrating the impact computational tools can have on design quality through requirements and change management. Shop-drawing revisions or change-orders based design modifications are common activities in the AEC field.

3.1 **CoM**

Design explorations and design optimization are among the common computational approaches that attempt to utilize design requirements as an integral part of the design activity. Design optimization views requirements as a fixed set of criteria and creates an evaluation function (referred to as the fitness function in artificial intelligence literature), which the design solutions are weighed against. However, design is seldom a frozen activity in time. Requirements as well as design solutions change as the search for the best design progresses. The co-evolutionary approach attempts model this dynamic relationship (Maher and Poon 1996). In Maher's approach the evaluation criteria, i.e. the fitness function, is not defined in advance, but is reformulated as an intermediate solution is reached. In this way, the requirement space co-evolves with the design solution. This model views design as a solution exploration and addresses the problem as a search algorithm.

The co-evolutionary model suggests two distinct problem search spaces in one of which requirements are specified and in the other the design solution is generated. Design progresses as these two worlds interact with each other. The movement in each space is viewed as an evolution, whereas movement between the spaces signifies co-evolutionary steps where either the problem leads to the solution or the solution refocuses the problem [Figure 4 a]. The solution space S(t) provides not only a state space where a design solution can be found, but it also prompts new requirements for

P(t+1) which were not in the original problem space, P(t)" (Maher and Poon 95). This model depicts an iterative system where requirements become an integral part of the design propagation.

While the CoM model separates the problem and solution spaces into two by introducing a process where each evolves in reaction to the other, design propagates with single operations. This single operation is specified by genetic algorithms through an evolving fitness function as a search process for solving a design problem. This model suggests a formal model of problem-design exploration.

As a computational model CoM would not have produced the faulty design solution change of the Kansas City Hyatt Regency Hotel. The fitness function in this case would be the optimum gravity load distribution of the hanging walkways. As new alternatives are searched each new design solution would have to satisfy the correct load distribution. In this particular case the CoM model also has the potential of detecting the fault in the original design, since the load factors of that one would also not meet the requirements. The revised broken cable detail would have been pruned from the search space before it made its way into the potential likely solution alternatives. In this respect CoM serves as a plausible approach to specifying requirements for change management through the life cycle of design.

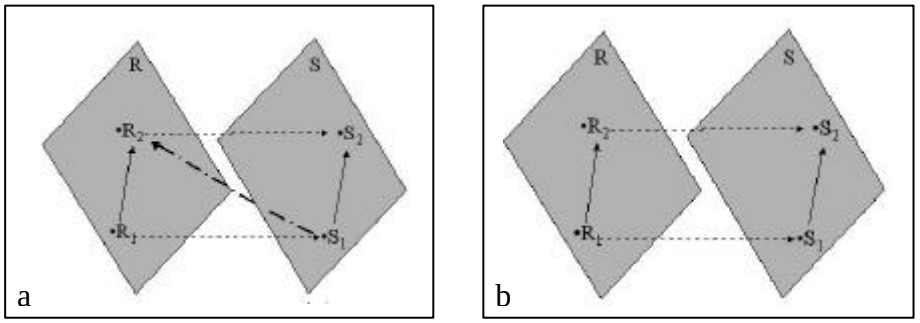

Figure 4 : a) **CoM where requirements (R) and solution (S) spaces are independent but affect each other through cross operations. b) MDM where R and S interact in only one direction**

3.2 **MDM**

In MDM the designer treats the requirements and solution domains independently. The requirement serves as a command for the implementation. Requirement-solution domains are parallel spaces with independent operations from each other. Traditional prescriptive design processes and theories follow this conventional model of design requirement specification.

There is no feedback from the solution domain to design in this model since

requirements and design explorations progress independently of each other. On the other hand, this model enables the refinement of requirements and independent search of alternatives in requirement and solution spaces. The pros of being able to refine and search in different domains is the ability to analyze and revise requirements without being influenced by solutions while they are still in their incomplete stages. Since there is no continuous feedback and re-factoring between two spaces, changes occur, as the once in the Kansas City Hyatt Regency case, as a result of such models. This does not mean that a multiple-domain, one-directional model does not provide a promising computational model. It suggests that the model should be coupled with other representations for completeness.

3.3 **SDM**

We suggest SDM as the most natural way the designers view design problems and approach solution domains. Each design decision and solution that comes with it serves as a new requirement to the next stage of design. the design problem not only becomes redefined, but it in fact becomes a new design problem within itself. Requirement and solution worlds are not separated from each other, they propagate in the same state space. Design requirements, which act as performance and geometric constraints, are not predefined and analyzed, yet they are considered as design evolves [Figure 5].

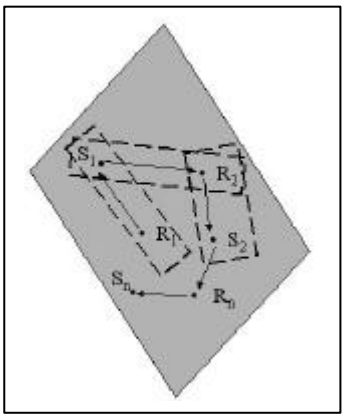

Figure 5: **In SDM requirements and solution exist in pairs in the same design space. Solution in one phase serves as a requirement to the next**.

There are two distinct ways designers seek solutions:

a. *Horizontal Design*: This is a stage where the designer has reached a certain understanding of the problem in design and is searching for best alternatives to his definition of the problem. Many well known computational algorithms for this common design activity have been proposed in generative layout managers as different search algorithms, constraint solvers and optimizers. The changes that occur in the parameters serve as requirements to each new search step

b. *Vertical Design*: For a design problem to be completed, the designer takes the problem through different stages, at each specifying and solving the problem in greater depth. Each vertical stage is a horizontal design problem within itself. However, design requirements apply to the overall design. With each design decision made the problem takes new shape while the requirements get exhausted.

The SDM takes into consideration both horizontal and vertical design iterations, therefore it deals with dynamic entities. Each operation moves the design along both axes. Both give room for creativity and consider design requirements as the steering force in design.

Due to the variations in context, client, purpose and technology used, every building design is unique. Similarly, there are different starting points for capturing requirements. Each alternative variation of the end product might be derived from different requirement priorities. Both abstract and quantitative requirements must be handled to ensure a seamless product development process, enabling backtracking and iterative design improvement along the entire process. Dynamic entities form as a result of a cyclic processes where design solution and requirements couple [Figure 6].

In SDM the cyclic process serves as a mechanism where design solutions are both checked against the initial requirements. They also serve as the new requirement sets for later design stages. In this respect, the changed rod design of the Kansas City walk way would initially be checked against the initial load factors. Moreover, as new solutions are developed, they would impose new specifications on the rest of the design, e.g. the solution necessitates the roof to be rechecked. Then this solution would become an input into roof design as a requirement. If the fault had not been detected earlier, then through this process, it would have to be detected later on.

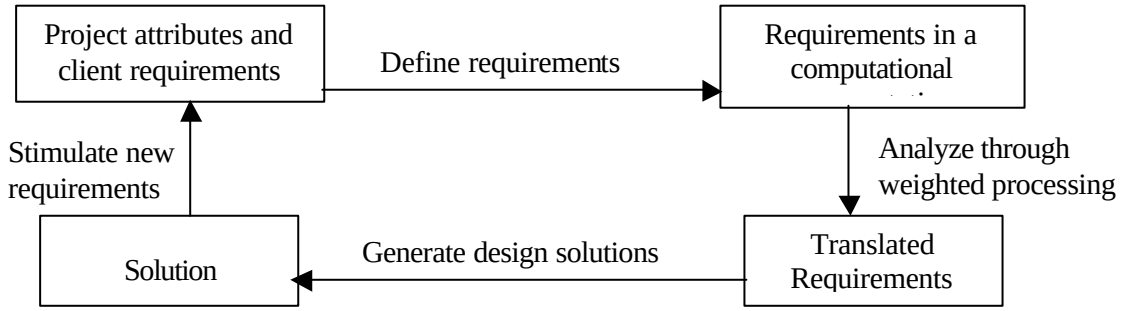

Figure 6: **State process diagram of SDM**

4 A COMPARATIVE VIEW

In this section we will compare and contrast each of the models discussed above towards fulfilling the characteristics that a computational environment for requirement specification should possess:

- *Formality*: It must be possible to represent a specification method in a

mathematically sound way. This is important in order to build computational models, re-use prior specifications and automate specifications for consistency checking and iterative design

- *Constructability*: What concepts are captured by the specifications and in which manner should be identifiable. For example, for spatial allocation, while the width-length-height tuple specifies a space volume, the same can be captured by a volume attribute, possibly based on an HVAC design consideration, where parameters are expressed as cubic volumes.
- *Minimality:* The model should impose the minimal number of parameters to define and specify an entity in the solution or requirement spaces.
- *Extensibility and Derivability* A model presents multiple views of the same data set. Thus, in a well-formed system, it should be possible to arrive at the complete model from different sets of specifications. This set of specifications can also be viewed as a model mapping functionality inherited in requirements specification. The multiple model views can be organizational hierarchy models, information, network models, spatial allocation models (equipments, furniture, etc.), cost driven models, and occupancy models [Figure 7].

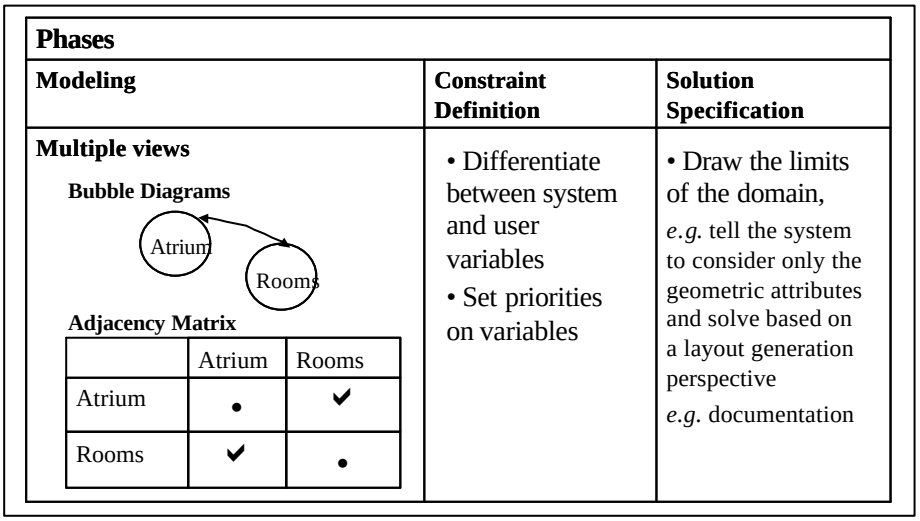

Figure 7: **Extensible and derivable representation of design requirements**

All three models suggest formal ways of representing requirement. SDM and CoM have greater potential in terms of constructability since the former re-iterates on solutions by redefining them as requirements and the later provides a fitness function for each design exploration. MDM on the other hand presents requirements as a large set where local explorations require greater effort to undertake. All three models are questionable in terms of minimality. The computational representation of design is a

significant research problem. SDM addresses minimality by suggesting a dynamic and cyclic requirement solution pair representation. SDM is the model in which extensibility and derivability can be easily tackled as a result of the solution domain being re-introduced into the problem as the requirement domain. Table 1 summarizes how the three models compare.

	CoM	MDM	SDM
Formality	+	+	+
Constructability	+	-	+
Minimality	-	-	-/+
Extensibility and Derivability	-	-	+

TABLE 1: **A comparison of the three models**

5 CONCLUSIONS AND FUTURE WORK

SDM spans both design progression and alternative generation activities through a dynamic framework. It provides a more plausible model for design problems in terms of fault and error detection and change management where requirement engineering is crucial.

CoM is a plausible model when a detailed quantitative evaluation function can be specified, but its applicability is limited when alternative search spaces need to be modeled.

MDM represents a more traditional view of design. Its treatment of the requirements and solutions as a mono-directional operation introduces complications and difficulties in reverse engineering. On the other hand, since it also treats each space in its entirety, it enables a more comprehensive analysis of each.

The future direction of this work will be to implement a prototype requirement-modeling application using SDM as the conceptual framework and visual modeling as the design driver.

6 REFERENCES

Akin, Ö. (1999) *Ethics and Decision Making in Architecture*, Unpublished Manuscript, School of Architecture, Carnegie Mellon University, Pittsburgh, PA 15213.

Akin, Ö, M. Donia, R. Sen and Y. Zhang (1995) SEED-Pro: computer assisted architectural programming in SEED, *Journal of Architectural Engineering* **1**, pp. 153-161.

Andriole, S. J. (1996) *Managing Systems Requirements: Methods, Tools and Cases,* Mc-Graw Hill, New York.

Cherry, E. (1999) *Programming for Design*, John Willer and Sons, New York.

Clayton, M. J., P. Teicholz, M. Fisher, and J. Kunz (1999) Virtual component consisting of form, function and behavior, *Automation in Construction* **8**, pp.351-367.

Hershberger, R. (1999) *Architectural Programming and Predesign Manager,* McGrawHill, New York.

Holtzblatt, K. and H. Beyer (1995) Requirements Gathering: The Human Factor, *Communications of the ACM* **38,** pp.31-32.

Imelinski, T., S. Navqi and K. Vadaparty, (1991) Incomplete objects - a data model for design and planning applications, *ACM Transactions* **2**, pp. 288-297.

Jackson, M. (1995) *Software Requirements & Specification*s, Addison-Wesley, New York.

Jacobsen I, Booch G, and Rumbaugh J. (1999) *The Unified Software Development Process*, Addison Wesley, Boston.

Kumlin, R. (1995) *Architectural Programming: Creative Techniques for design Professionals,* McGraw Hill, New York.

Leffingwell, D. and D. Widrig (2000) *Managing Software Requirements a Unified Approach*, Addison-Wesley, Boston.

Liskov, B and S. Zilles (1975) Specification Techniques for Data Abstraction, *IEEE Transactions of Software Engineering* **1**, pp. 7-19.

Maher, M. L. and J. Poon (1995) Co-evolution of the fitness function and design solution for design exploration, *Proceedings of IEEE International Conference on Evolutionary Computing, IEEE*, pp. 240-244.

Maher, M. L. and J. Poon (1996) Modeling design explorations as co-evolution, *Microcomputers in Civil Engineering* **11**, pp. 195-209.

Marshall, R. D., E. O. Pfrang, E. V. Leyendecker, K.A Woodward (1982) *Investigation of the Kansas City Hyatt Regency Walkways Collapse*, US Government Printing Office, Washington.

Nielson, J. (1993) *Usability Engineering*, Morgan Kaufmann, New York.

Pena W, S. Parshall and K. Kelly (1987) *Problem Seeking: An Architectural Programming Premier*. AIA Press, Washington.

Pidd M. (1996) Five Simple Principles of Modeling, *Proceedings of the 1996 Winter Simulation Conference*, pp. 721-728.

Sanoff, H. (1974) *Methods of Architectural Programming* Stroudsburg, PA: Dowden, Hutchinson & Ross.

Sanoff, H. (1992) *Integrating programming, evaluation and participation in design: a theory Z approach*, Avebury, Aldershot.