Data and Decision Support with Dynamic Representation of Multiple System Requirements Models for Building Design

Ipek Ozkaya
School of Architecture
Carnegie Mellon University

May 2000

Advisor
Prof. Omer Akin
School of Architecture
Carnegie Mellon University
# Table of Contents

1. Motivation ............................................................................................................. 1
   1.1 Introduction........................................................................................................ 1
       1.1.1 Practical Motivations ................................................................................... 1
       1.1.2 Case-based Reasoning (CBR) in Architectural Programming (AP) .......... 2
   1.2 Research Objectives.......................................................................................... 3
       1.2.1 Requirements Modeling of Building Products.......................................... 3
       1.2.2 Information Sharing in Building Design..................................................... 3
       1.2.3 Dynamic Programming ............................................................................. 4
       1.2.4 Data Representation in Dynamic Requirements Models ......................... 4
       1.2.5 Decision Support through Intelligent Agents.......................................... 5
       1.2.6 Generative Programming ....................................................................... 5
   1.3 Conclusion ......................................................................................................... 6

2. Scope ..................................................................................................................... 7
   2.1 Introduction........................................................................................................ 7
   2.2 Problem Statement........................................................................................... 7
   2.3 Definitions......................................................................................................... 8
       2.3.1 Architectural Programming ..................................................................... 8
       2.3.2 Architectural Programmer ...................................................................... 9
       2.3.3 Dynamic Programming .......................................................................... 9
       2.3.4 Requirements Modeling ....................................................................... 10
       2.3.5 Specifications ....................................................................................... 10
       2.3.6 Product Modeling ............................................................................... 11
       2.3.7 CBR ...................................................................................................... 11
       2.3.8 Computable Decision Support ............................................................. 12
       2.3.9 Data Related Terms .............................................................................. 12
   2.4 Conclusion ....................................................................................................... 13

3. Rationale of AP during Early Phases of Design.................................................. 15
   3.1 Introduction...................................................................................................... 15
   3.2 AP Process....................................................................................................... 15
       3.2.1 Design Rationale .................................................................................. 18
       3.2.2 Collaborative Design ........................................................................... 18
       3.2.3 Feasibility Studies .............................................................................. 19
       3.2.4 Functional Design Decision Support .................................................... 20
       3.2.5 Performance Analysis & Evaluation ...................................................... 20
       3.2.6 Reverse Engineering ........................................................................... 20
       3.2.7 Knowledge-based Design .................................................................. 20
       3.2.8 Multiple System Integration ................................................................. 20
   3.3 Scenarios from Practice .................................................................................. 21
       3.3.1 Sanoff’s Scenario [1992] ...................................................................... 21
       3.3.2 Kumlin’s Scenario [1995] ..................................................................... 21
       3.3.3 SEED Scenarios ............................................................................... 22
       3.3.4 Evaluation ........................................................................................... 23
   3.4 Conclusion ....................................................................................................... 23
4. Information Exchange & Building Product Models............. 25
   4.1 Introduction .............................................................................. 25
   4.2 Information Modeling in Computer-aided Design Tools ............ 25
      4.2.1 Architecture, Engineering and Construction (AEC) Industry .... 26
      4.2.2 Data Modeling Perspective .................................................. 27
      4.2.3 Universe of Discourse .......................................................... 28
      4.2.4 Database Support ............................................................... 28
   4.3 Similar Research ....................................................................... 29
      4.3.1 Shared Data Support ............................................................. 29
      4.3.2 Standardization Efforts ....................................................... 32
      4.3.3 Information Modeling Languages ....................................... 33
   4.4 Conclusion ................................................................................ 34

5. Application Domain ............................................................... 35
   5.1 Current System Architecture in SEED ....................................... 35
      5.1.1 The Architectural Programming Module ............................... 36
      5.1.2 Information Exchange through Shared Schema in SEED - SPROUT . 39
      5.1.3 SEED Database ................................................................. 41
      5.1.4 Agent Communication Model .............................................. 42
   5.2 Evaluation ................................................................................ 43
   5.3 Prototype Problem Design ....................................................... 44
      5.3.1 Generating a communication model in SP-II ......................... 45
   5.4 Implementation Design ............................................................ 47
      5.4.1 Factories ............................................................................. 47
      5.4.2 Handlers ............................................................................. 49
      5.4.3 Mapping Manager ............................................................... 52
      5.4.4 Language Bindings ............................................................... 53
   5.5 Conclusion ................................................................................ 54

6. Preliminary Proposed Framework ........................................ 55
   6.1 Introduction .............................................................................. 55
   6.2 Context .................................................................................... 55
   6.3 Domain Study ............................................................................ 56
      6.3.1 User Space: actors ............................................................... 56
      6.3.4 Specification Space ............................................................. 57
      6.3.1 Usability Goals of the Decision Support Module .................... 57
      6.3.2 Core Features ................................................................. 59
      6.3.2 Software Engineering Patterns ........................................... 60
   6.3 Use Case Explorations ............................................................. 61
      6.3.1 Libraries - A CBR abstraction ............................................. 61
      6.3.2 Generation Mechanisms ..................................................... 64
      6.3.3 Product Modeling ............................................................... 64
      6.3.4 Specification Units ............................................................. 65
   6.4 Conclusion ................................................................................ 66
7. Conclusion

7.1 Contributions

7.2 Future Research

References
List of Figures

FIGURE 2.1. General architecture of SEED [http://seed.edrc.cmu.edu/IJDC/] ...................... 8
FIGURE 4.2. ICADS System Software Architecture: [Myers et. al 92] ............................... 30
FIGURE 4.3. EDM-2 : [Eastman & Jengs 99] ................................................................. 31
FIGURE 4.4. GARM product model representation decomposition: [Tolman 99] ............. 32
FIGURE 5.5. SEED Module Generic Architecture: [Flemming & Woodbury 95] .......... 35
FIGURE 5.6. SP-I’s project structure: [Donia 98] .......................................................... 36
FIGURE 5.7. SP-I’s building category specifications model: [Donia 98] ....................... 37
FIGURE 5.8. SP-II’s product modeling object model: [Donia 98] ............................... 38
FIGURE 5.9. Design Requirements Object Model in SP-II: [Donia 98] ...................... 39
FIGURE 5.10. General Information Exchange System Architecture in SEED .......... 40
FIGURE 5.11. Agent Connection Architecture in SEED: [Flemming et. al. 00] ........ 42
FIGURE 5.12. SPROUT Compiler: [Flemming et. al. 00] ............................................. 43
FIGURE 5.13. Seed Prototype Implementation Library ............................................... 46
FIGURE 5.14. Test SP-II Models .................................................................................... 46
FIGURE 5.15. Implementation modules for integration ................................................. 47
FIGURE 5.16. General representation of factory method: [Gamma et. al. 95] ............. 48
FIGURE 5.17. SeedProFactory, a generic approach ...................................................... 49
FIGURE 5.18. Handlers: [Gamma et. al. 95] .................................................................. 50
FIGURE 5.19. SeedPro Object Handlers ...................................................................... 51
FIGURE 5.20. Domain handlers .................................................................................. 52
FIGURE 5.21. SPMappingManager .............................................................................. 52
FIGURE 5.22. SP-II Generic Language Bindings Model ............................................ 53
FIGURE 6.23. Domain Study ....................................................................................... 56
FIGURE 6.24. Builder Pattern: [Gamma et. al. 95] ....................................................... 60
FIGURE 6.25. Model for library layers, in OMT notation ........................................... 61
FIGURE 6.26. Integration Sequence of Library Instantiation, using Rational Rose .... 61
FIGURE 6.27. Synchronous collaborative modeling - DB interaction ...................... 65
FIGURE 6.28. Specification Unit Knowledge-Base Interaction .................................... 66
1. **Motivation**

This section provides a background on the motivations of this study along with a background on what is seen as missing in computer-aided design (CAD) studies supporting information exchange with generic requirements models of building design.

1.1 **Introduction**

What initiated this research of data support with dynamic representation of multiple system requirements models for building design was the database maintenance needed in the Architectural Programming (AP) Module of the Software Environment to Support Early Phases of Building Design (SEED) project, Seed-Pro (SP-II).

Lack of computable architectural design decision making tools, motivated the extension of the problem to include case-based reasoning (CBR), which depends on persistent database support.

The product of the work is the reflection of the information gathering, processing, and problem definition and design process.

1.1.1 **Motivations**

The studies conducted on AP by Akin demonstrate that although AP is a dominating part of the design process, lack of computational tool support and formalized processes cause information generated during AP to be lost. Some other findings of the study are [Akin et. al. 1995]:

1. The program serves as a complete inventory of design requirements and criteria of design evaluation.
2. The program is an informal means for resolving initial budget, design, client and other resource limitations.
3. Documenting the program is an iterative process, however after the formation of functional design the architectural design drawings replace the program, thus the program becomes outdated.
4. In profession there are inefficient attempts to reuse past programs.
5. Developing the architectural program is a task of a team with diverse expertise.

All the findings sited above suggest that the computational tool development for AP as an extent of requirements modeling must be accompanied by consistent, semantically correct, extendable and robust data support, hence a database. It has been accepted that CAD uses design databases in many ways like building knowledge-based reasoning problem solving modules, providing integrated design across different applications and domains, facilitating collaborative design and decision support. I address

- building the database connection of the AP module, SP-II\(^1\), of SEED [Flemming et. al. 95] (a detailed description is provided in Chapter 5) and
- building a dynamically defined data framework for AP addressing requirements modeling issues in architecture, engineering and construction (AEC) industries. Such a framework can be utilized to offer a robust knowledge representation and organization base towards computable intelligent decision support (DS).

1.1.2 Case-based Reasoning (CBR) in Architectural Programming (AP)

In the past few decades, both computational and cognitive paradigms of CBR have been tackled in various domains such as medicine, structural engineering, management sciences, hardware development, etc. [Smyth ed. 98, Smith ed. 96]. Case-based Design (CBD), as a design process of using precedents is also not a new paradigm in the realm of architectural design [Oxman 94]. However, the practice of architecture in this context is usually limited to the functional design stage when the architect has solved some of the preliminary problems or has been given the requirements that come out of the early phases of design, as AP.

SP-II addresses the computational support problem during the AP stage of design [Akin et al 95]. The need defined through SP-II for AP is based on the long ongoing research of ill-defined, multi dimensional, and interdisciplinary nature of requirements capture in building design problems. However, in terms of AP, in more general requirements modeling view of the design process, the computable information management and knowledge representation have not been fully discovered to assist DS.

Capturing the data produced and processing it for future reference in AP is an area of study with potential benefits to the profession.

1.2 Research Objectives

Data support for dynamic representation of requirements is a problem which serves many sub research areas that are important for enhancing the field of architectural practice, education and research.

1. Several prototypes for the domain of AP have been developed at the Department of Architecture at Carnegie Mellon University by O. Akin, M. Donia and R. Sen. This study uses the latest prototype SP-II, which employs a product modeling abstraction to AP [Donia 98]. Comparisons with the others will also be made.
1.2.1 Requirements Modeling of Building Products

It is possible to view design as an overloaded information processing, in which the information necessary for “production” is generated, retrieved and modified. This information contains some basic quantitative data, which are also constraints or requirements in terms of the production process. Such constraints and requirements guide the design process and shape the manufacturing environments like form features, proximities, adjacencies, dimensions, performance related issues as thermal acoustical or lighting expectations, etc. [Pham 91].

The product of programming is the statement of the problem, which may be used as requirements model representation to guide design. In terms of a tool where AP information is processed the task becomes, generally speaking, the generation of the requirements model that utilizes this information as a part of the design process towards production.

Thus, AP information carry requirements and specifications on the properties of the product providing the standardized and constraint based support. Computational tool development must take into consideration the integration of multiple expertise domains which generate such constraints and standards as requirements.

1.2.2 Information Sharing in Building Design

The nature of building design is interdisciplinary, iterative and requires management of multiple levels of information in different stages of the design process. The information created at one stage along the process of design becomes input and constraint to the later stages. The decisions made by the team must be communicated and coordinated. This necessitates:

- Seamless integration of different processes in design through the information generated
- Platform independent sharing of data to facilitate generalization
- Creating shared and distributed data support to backup specialized applications for building design
- Synchronization of information in distributed, asynchronous and heterogeneous design agents

In building design, requirements specification in the form of an AP could become a medium to capture formalized, common information modeling to base the decisions on.

The constraints and specifications that are communicable to the different agents along the process support the synchronized information sharing. AP as a view of requirements modeling should be designed to support multi domain information exchange.

1.2.3 Dynamic Programming

Object-oriented (OO) and human computer interaction (HCI) based software development models are the dominating trends of 90s and 00s in application development. The aim is to come up with general models which will meet multiple user needs.

While generic approaches are plausible in solving the problems of predefined and agreed upon domains, for industries as AEC they do not always give productive results.
Motivation

All inclusive models generated for the use of building design have proven to be inefficient and unsuccessful [Snyder & Flemming 99], [Donia 98], [Eastman & Jeng 99], [Bjork 91]. Therefore, while dynamic programming approaches are headed in the correct direction, there are issues which have to be addressed in terms of consistent information management to improve DS.

Dynamic programming as a software development approach is a medium to exemplify and improve how multiple users could utilize the same tool in different subdomains.

1.2.4 Data Representation in Dynamic Requirements Models

AP as a requirements model comprises a set of specifications that are attribute descriptions of the building to be designed. Donia prescribes the computer representation for modeling building design requirements as fulfilling two main goals:

1. To create a repository of design requirements that contains all the information the designer needs to specify
2. To support extracting information from that repository and communicating it to the design systems that collaborate on the process of producing a building design.

The construct employed in realizing these goals in SP-II is the support of definition of multiple conceptual schemata, and representing different ways of describing design specifications from different views. The schemata categorizes information needed to create design requirements for a certain building product [Donia 98].

The storage and retrieval of multiple schemata resulting from different requirements views of a building product has to resolve issues of:

1. uncontrollable growth of the database and creating scattered and redundant information that the users cannot benefit from
2. semantically correct retrieval and navigation of the data from the user perspective
3. mapping of the requirements data into the database and the computational overhead of the process
4. the necessity to communicate, propagate, and update changes without computational overheads in needs of modifications as a result of the iterative nature of the design process
5. providing data usage across different product schemata produced towards intelligent agents laying over database system architectures.

1.2.5 Decision Support through Intelligent Agents

It is often argued that the transfer of knowledge in ill-structured domains should be based on case-based representation [Muller & Pasman 96]. In artificial intelligence, CBR approaches offer a methodology that specific instances of previously encountered problems, hence knowledge represented in those problems, are used instead of generic rules or grammars. Successful cases of previous instances create an evaluation framework for the new problem contexts.

The identification of the kind of knowledge to be captured in previously encountered problems of building design requirements model also formulate the form of DS. In terms of AP knowledge typology representations are also possible. Typology problems
suggest generic approaches to a knowledge-base which grows as variations on the same basic rules are provided.

In order to acquire flexibility to assist ill-structured problem domains, like design, each complex case must be decomposed and represented along many partially overlapping dimensions. The same information can be retrieved in multiple ways instead of retrieving a precompiled schema. Situation specific schemes have to be derived depending on the nature of the new problem context. A consistent database which supports management of multiple schema in a shared database provides a well structured foundation for building knowledge-based applications for DS.

The re-use of APs in future problem contexts is not a formally established process. Yet, the re-use of precedent designs is a generally accepted fact in the design studies research [Akin et. al 1996].

Re-use of requirements models have promising contributions to building design as:

- Rapid generation of initial feasibility reports based on past similar realized cases
- Generation of new programs based on precedents
- Typological classification of APs into categorizational knowledge-bases
- Integration of changes into the cycle of building design in a more rapid way

Studies have already proven that APs are re-used in the profession, yet in a novice manner that does not benefit from computer support. Generation of software tools which take these into account could serve this need.

1.2.6 Generative Programming

The purpose of requirements modeling is to ease production and design, consistency check and evaluate the end product by aiding in generation and derivation of functional requirements and design units. The specifications provide the framework on which such generations and alternatives can be based. These are reasoning mechanisms that are employed in the design process.

In AP, derivation of spatial requirements is the most common generative programming motivation. In SP-II this question was addressed by Sen [1998]. Sen proposed the use of technology construct which in SEED is defined as a collection of computational mechanisms that creates, details, and instantiates design and functional units satisfying the requirements of a class of functional units in a design context based on specific construction technologies or form generation principles [Woodbury & Fenves 94]. As the range of the technologies to support generative programming compositional, decompositional, mapping, dependency and evaluation and reasoning mechanisms are proposed.¹

Information encapsulation, representation and re-use towards DS should employ the capturing of such reasoning mechanisms of generative programming notions. In

¹. Currently in SP-II some low level generation mechanisms to support generative programming exist, like basic formula generation to derive FU attributes. A work in progress also addresses the issue of generative programming by focusing on derivation of functional requirements from non functional specifications by H. Erhan.
Motivation

dynamic requirements modeling the generic mechanisms employed do not only provide decision alternatives but they also suggest design process and evaluation constructs for future reference. Data support must also take into account the management of dynamically defined generation constructs and facilitate the re-use of such knowledge seamlessly across different requirements models.

1.3 Conclusion

The problem domain involves integrating a number of core research areas as design processes and design decision making [Akin 86], knowledge-based reasoning [Kolodner 93], human computer interaction [Dix et. al. 98], design databases [Koutamis et. al. 95], product modeling in AEC [Bjork 91, Eastman& Jeng 99], OO and dynamic software engineering [Gamma et. al. 95, Pree 95].

The focus of this study is on data management as an integration medium of the these research areas in requirements modeling with special emphasis on implementation specific issues in the context of SEED.
2. Scope

Prior to describing the application specific design issues it is beneficial to set the limits of the domain that is referred to. This section aims to describe the scope through problem statement and definitions related to the computational application and AP.

2.1 Introduction

The scope of the problem in this stage as a part of SEED is mostly application driven in the context of the SEED modules’ information exchange through the database. However, as later in Chapter 5 it will be described in detail, there are conceptual mismatches in terms of the prescribed SEED modular agent system architecture, product modeling concepts, and dynamic programing. Therefore, the domain and its data support requirements should be understood to be able to critique the current system into matching it with the needs of a dynamic platform.

2.2 Problem Statement

The problem is an application and implementation driven problem to assist the requirements modeling domain exemplified by AP in the context of SEED. Figure 2.1 summarized the general modular architecture of SEED.

The goals of SEED are described by Flemming and Woodbury as:

“...This includes using the computer not only for visualization, analysis and evaluation, but also more actively for the generation of designs, or more accurately, for the rapid generation of computable design representations describing conceptual design alternatives and variants of such alternatives with a sufficient level of detail that enables sophisticated evaluation tools to receive all of the needed input data from the representation”. [1995]

A database supports SEED in order to help achieve the above goals in multiple ways:

- by storing temporary design versions and design variants within a specific project
- by storing cases that can be reused across projects
• by storing projects to be used across different modules

FIGURE 2.1. General architecture of SEED [http://seed.edrc.cmu.edu/IJDC/]

The modules are not intended to function in a linear sequence. The underlying idea is to be able to capture the iterative nature of the design process through such a system architecture by the support of a database [Flemming et. al. 1995] Therefore, the database is the media for information exchange which should incorporate the conceptual initial motivations of SEED like dynamic, generic, and interactive problem specification. The software architecture pursued should integrate dynamic and generic programming techniques.

The purpose of the study is to establish the above database goals along with the requirements and goals of the SPII module.

2.3 Definitions

The terms assumed in the context of SEED in the overall, the AP Module SP-II and this research are defined below.

2.3.1 Architectural Programming

*Programming is generally viewed as an information processing system setting out design directions that will accommodate the needs of users, the client, the designer or the developer [Sanoff 92].*

*Architectural programming is the research and decision making process that defines the problem to be solved by design [Cherry 99].*
As a plan of action AP existed as long as architecture. Familiarity with the building type, and design and construction occurring simultaneously until recent history made AP seem not really occurring although it was just not in the format that we today comprehend. Its necessity and importance became more evident with the development of fields such as psychology and sociology, the interdisciplinary nature of the profession was better understood and excepted, and more complex structures were needed.

The intent of an architectural program may be a new building, orderly operations or facilities growth, improved operational efficiency, better working environment or informed choices of site location. Usually a building program is merely taken as an AP, whereas the former is just an aspect of the later. A program for a new building serves as a tool for recording client needs, insuring that these needs are met and evaluating the building design before construction begins [Pena 87].

AP through a program concept, an idea at the lowest level of abstraction for a program, assist the derivation of a design concept which suggests a way that the program can be implemented. Therefore, the extends of an AP is beyond an initial agreement between parts, but it is also what the alternative end products refer to.

2.3.2 Architectural Programmer

The problem solving process of an architectural programmer may or may not be similar to that of an architect. The definition of an architectural programmer has to be made because this eventually describes the users of SPII, hence the users of the data. William Pena differentiates between architectural designers and programmers as:

“Do designers program? They can, but it takes highly trained architects who are specialized in asking the right questions at the right time, who can separate wants from needs, and who have the skills to sort things out. Programmers must be objective (to a degree) and analytical, at ease with abstract ideas and able to evaluate information and identity important factors while postponing irrelevant material. Designers can't always do this. Qualifications of designers and programmers are different” [Pena 87].

2.3.3 Dynamic Programming

Dynamic programming implies a run time (not compile time) determination of the methods that is to be called. [Pree 95]

For a problem to be solved using dynamic programming techniques it must allow being divided repeatedly into subproblems in such a way that identical subproblems arise again and again. Dynamic programming involves systematically recording the solutions to these subproblems to be reused in solving the entire problem. It is an approach developed to solve sequential decision problems [Jacobsen et. al. 99]. While it gives the application the flexibility to serve the user domain with generic customizable environment, the data support has to be designed accordingly.

The determination of size, type and methods to be employed in data structures in run time in contrast to the static approach where all these are predetermined in compilation has a lot of advantages in the object oriented software development domain as:

1. Ensure memory management towards higher performing tools.
2. Support generality and need-based determination of data structures to support multiple types of data
3. Support multiple views of the domain models in the application

Making computer-aided design tools easier to use is one of the main interest and boom in interactive and dynamic approach in software development.

2.3.4 Requirements Modeling

For generations, certain Native American tribes built a kind of canoe, called a dugout, made of a hollowed-out log. The canoe builders began by looking for a tree that was several feet in diameter that had already toppled over near the water. Near it they lit a fire and spread the hot coals on the top of the log. The charred wood was much easier to hollow out with stone tools. After several days of carving, the canoe would appear to be complete, and the builders would push and pull it into shallow water. More than likely, the first rough effort simply rolled over. It was not balanced. More work with those dull stone tools followed, until they had a boat that did not capsize when someone bent over to pull a fish out of water. Only then did they call it finished.[Jacobsen et al. 99]

The word requirement is usually used in a general sense meaning “needs”. These needs are not necessarily understood entirely, which leads to the necessity of the emergence of the concept “requirements modeling”.

An important task in requirements modeling is the discovery of requirements capture; i.e. the process of finding out what best describes what is to be built. This process is beyond expecting the clients to know what they require, that is only a part of it.

Every building is unique as a result of the variations in context, client, purpose, technology used and so on. Similarly, there are different starting points for capturing requirements. For each alternative end product even in similar contexts might be derived from different requirements hierarchies and priorities. Both abstract and quantitative requirements must be handled to ensure seamless product development, backtracking and iterative design improvement along the design process.

2.3.5 Specifications

...for example, a requirements specification is a document containing requirements; a design specification is a document containing the design; a test specification is a document containing the testing process [Davis 90]

...every specification is, in a formal sense, a requirement.... Specifications are difficult things. Although they are requirements of a kind, they may not seem to make obvious sense in the environment.[Jackson 95]

The distinction between specifications and requirements in an important one to draw in the domain of AP because an AP comprises both concepts in different levels of abstraction. AP is a requirements specification.
2.3.6 Product Modeling

A product model as a general concept is a conceptual description of a product, capable of structuring all the information necessary for the design, manufacturing and use of that product. Rather than a single database, a product model should be viewed as a common language for the description of a particular type of product or as a more complex form of traditional classification system. The model or schema can then be implemented in slightly different ways in different application programs. [Bjork 91]

Product modeling is the representation of part of structures towards generic product structures describing multiple different but related structures in terms of parameters that reflect their descriptive and performance characteristics. It is an industry driven, prefabrication and pre-engineered based approach to increase market competitiveness through rapid customization of products according to client requirements. With a product modeling approach while the price and time spent in production is optimized the quality is enhanced. The inheritance of the approach for buildings as products also has its roots in the boom of AEC becoming established industries.

The product model should:

- be comprehensive,
- cover the information created during all the stages of design and manufacturing process,
- not contain redundant data,
- specify what information contained in the model, not how it is stored in the computer.

All of the above is information captured in AP.

A product model view of a building in the form of AP should combine descriptive, performance and context parameters. Descriptive parameters, as geometry, color, etc., are those controlled by the decision maker. Performance parameters, as comfort levels, energy requirements, etc., are those that decision makers use to judge the appropriateness of a product. Context parameters are those that used to describe the environment within which the product is assumed and evaluated. [Papamichael et. al. 99]

2.3.7 CBR

In a CBR oriented tool the user solves new problems by remembering previous occurrences similar to the new problem. The extends of case based reasoning can mean adapting old solutions to solve new problems, using old cases to test new solutions, using old cases to interpret new situations or using old cases to explain new situations. As a basic rule several cases are needed to solve complex problems. The main challenge in designing a productive case based system is in being able to define the cases of the domain correctly [Kolodner 93].

As a process model the CBR involves several operations:

- recall relevant cases from case base (case memory, e.g. a database)
- select the most matching example,
• adapt it to the new problem,
• test,
• evaluate results,
• update case-base with the new case.

The researches in many other areas that concern case based reasoning approaches to their applications domain all try to answer the following questions to be able to come up with solutions.

• What makes up a case? How can the case memory be represented?
• How is memory organized? What are indexing rules?
• How does memory change? How do the case memory and indexing rules change over time?
• How can old solutions be adapted to new situations? What are the similar metrics and modification rules?
• What are repair rules?

2.3.8 Computable Decision Support

The key to the effective application of computational technology is to conceive, model, design and evaluate the joint human-machine cognitive system. Like Gestalt principles in perception, a decision system is not merely the sum of its parts, human and machine... The joint cognitive system paradigm demands a problem-driven, rather than technology-driven, approach where the requirements and bottlenecks in cognitive task performance drive the development of tools to support the human problem solver. As a result, computational technology should be used, not to make or recommend solutions, but to aid the user in the process of reaching a decision.[Woods 85]

Information retrieval is just a part of computable decision support. The essence of decision support through a computable environment is the extend in which the users are enabled to

• achieve goals,
• access alternatives for achieving the goals,
• make relations between alternatives and goals in order to choose the best alternative.

2.3.9 Data Related Terms

The data related terms which are repeatedly referred to in this document are used in the meanings given below [McFadden et. al. 99]

Database.

A database is an organized, that is structured so as to be easily stored manipulated and retrieved by users, collection of logically related data of any size and complexity
Data. 
*Known facts that could be recorded and stored in manual or computerized media. Data consists of facts, text, graphics, images, sound, video segments that have meaning in the users’ environment. Data becomes useful only when it is placed in some context.*

Information. 
*Data that has been processed in such a way that it can increase the knowledge of the person who uses it.*

Metadata. 
*Data that describes the properties or characteristics of other data, like data definitions, data structures, rules, and constraints.*

User view. 
*A logical description of some portion of the database that is required by a user to perform a task.*

### 2.4 Conclusion

This study has to consider a number of different research areas in addressing the integration problem posed in SEED and extending it to computable intelligent decision support in SP-II. This chapter set the boundaries of the agenda by explaining the scope of some of the terms which are extensively referred to throughout the paper.
Scope
3. Rationale of AP during Early Phases of Design

This study assumes a central role for AP with a dynamic requirements modeling approach in early phases of AEC design. It is believed that a well formed requirements environment allowing constraint management, consistency checking, collaborative scenarios, decision inferences and generative capabilities has potential in advancing the use of CAD tools for design. This section therefore is devoted to provide a background discussion on the rationale of AP during the early phases of design.

3.1 Introduction

How should programming be done to achieve architectural excellence, how can programming help the built environment be more responsive to the user’s needs and the needs of the environment, how should it be computed are some of the questions that should be addressed along with data support.

AP in the overall design process is wrongly viewed as a prelude to design, being completed prior to its beginning. My approach is considering AP as continuous and simultaneously occurring with design throughout the process.

Programming is

- a form of design,
- prescriptive management and
- an iterative process.

Computable features of AP process is inherited in understanding the requirements management as complementing the problem solving involved in the quality of the building products.

3.2 AP Process

For an application platform to assist computable AP in the design process the procedures of the AP process that are computable must be identified. The information and data generated through such functionalities would determine the design of intelligent decision support and knowledge-base agents on top. The computability of design should not aim to imitate the process as it would happen, yet it must sieve those functionalities
that are more beneficial in a computable platform and design the data support accordingly.

Kumlin describes some of the criteria along which to judge the success of an AP as

- Were the program predictions regarding scope and cost achievable and accurate?
- Was the predicted efficiency achieved and did the final design meet the expectations of the client regarding the quality?
- Did the program allow the widest latitude for creative design and simultaneously keep the final result within the quantitative parameters?
- Was there enough information, and was it organized and expressed in a manner to minimize information overload and be easily accessible?
- Were the client’s requirements, visions, dreams, intentions, and priorities clearly and immediately apparent to the design team and manifest in the final solution?
- Did the program have the support of decision makers? [1995]

In a study Sanoff conducted on varies models of AP among those established in literature he identified a set of common procedures and which models incorporate these procedures into the AP model they suggest. Table 3.1 gives a summary of his findings [1992].

<table>
<thead>
<tr>
<th>Procedural Steps Suggested</th>
<th>Davis</th>
<th>Farbstein</th>
<th>KMD</th>
<th>Kurtz</th>
<th>Moleski</th>
<th>Pena</th>
<th>White</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Plan The Program</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identify participants &amp; organize programming team</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Identify programming objectives</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Identify programming context</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Identify information needed</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Define with client - process, sequence, tasks, schedules, rules, responsibilities</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Identify primary information sources</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><strong>Understand Client’s Organization &amp; Philosophy</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nature of organization, image &amp; philosophy</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Organization function &amp; communication process</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Satisfaction &amp; dissatisfaction with present facility</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Identify user objectives</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><strong>Establish Project Goals</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional goals</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Form related goals</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Economic goals</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Time related goals</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><strong>Organizing Information Search</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collect &amp; organize project related facts</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
### TABLE 3.1. Comparison of procedural steps suggested by different researchers in AP: [Sanoff 92]

If the above list is studied in detail it is seen that some of the procedures suggested display a common understanding in the practice. Some of these might not require a computable platform with extensive data support, yet might benefit from it, while others would require such a tool to improve the practice. The study of the application domain must take this into consideration. The use-case study of Chapter 6 will refer back to the above procedures.

<table>
<thead>
<tr>
<th>Procedural Steps Suggested</th>
<th>Davis</th>
<th>Farbstein</th>
<th>KMD</th>
<th>Kurtz</th>
<th>Moleski</th>
<th>Pena</th>
<th>White</th>
</tr>
</thead>
<tbody>
<tr>
<td>Questionnaires &amp; interviews</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Literature search</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Observation of existing operations &amp; facilities</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background information from client</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review similar building types &amp; operations</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Collect facts related to building function</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Collect facts related to building form</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collect facts related to building economy</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collect facts related to time</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td><strong>Analyzing Information</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyze collected facts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional space standards</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tabulation of space requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spatial diagrams</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interaction patterns among activities</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Written description of functional units</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Conceptual Development</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncover, test &amp; develop conceptual alternatives</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Develop functional concepts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop form, economy &amp; time related concepts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td><strong>Identify Budget related Problems &amp; Needs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional needs</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Form needs</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Economic needs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td><strong>Project Impact</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Impact on client’s organization &amp; operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Community &amp; ecological impact</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program review and revision</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If the above list is studied in detail it is seen that some of the procedures suggested display a common understanding in the practice. Some of these might not require a computable platform with extensive data support, yet might benefit from it, while others would require such a tool to improve the practice. The study of the application domain must take this into consideration. The use-case study of Chapter 6 will refer back to the above procedures.
A computable platform to support AP as a requirements model should not be a documenta-
tion tools, there are already many generic tools which could give such support. It
must incorporate information processing, decision inference, and knowledge accumu-
ation.

3.2.1 Design Rationale
Regardless of whether programming is considered separate from or integral to design
the intent of each programming model is to outline categories of information to be col-
clected, analyzed and organized to assist in the design of a facility.

AP is a process of problem identification, information collection and information orga-
nization resulting in a communicable statement of intent. It is a way of systematizing the
design process.

Design evolves around ill-defined and ill-structured processes, especially during the
eyl phases [Akin et. al. 96]. A requirements modeling approach as a design rational is
an attempt to formalize this fuzzy domain. In the context of requirements modeling
some software engineering design principles are relevant and can be employed to AP.

Requirement dependency as a design rationale captures required behavioral dependency
relationships between individual components behaviors and specifications and how
such dependency relationships are realized by the elements of the AP. Dependencies
such as where one component or subsystem causes a behavior in another component or
subsystem and where a specific order of relationships have to be maintained have to be
captured in design. The requirements model should be flexible enough to employ such
capabilities in data. Inconsistency dependency as a design rationale on the other hand
addresses the refinement of inconsistencies or conflicts that arise. [Bose 98]

3.2.2 Collaborative Design
AP supports design collaboration in two distinct levels, both of which must be recog-
nized in a computational tool to support the domain. First of all, creation of an AP is the
task of an interdisciplinary team which comes together with different expectations from
the AP to be realized. The team includes all the people that have power to effect change
either by order or influence and people who serve as information resources. The cre-
ation of the AP as the end product necessitates the agreement and collaboration of this
team. AP is seldom the task of one person.

The process of AP creates both a document and the environment in which it is to be
employed. Kumlin in describing architectural programming as a part of design design-
ates the process explicitly as a one involving iterative meetings towards making deci-
sions in creating an AP [1995]. Thus, to capture both the process and the goal of AP the
computational tool must take into account the informal information exchange that takes
place among the people involved. As it later will also be explained in detail the media
that such phenomena take form in the computational platform cannot be mimed and
mapped as it is in real life, however the alternative must be comprehensive enough to
support the process. In terms of computation, the data representation and database sup-
port is just a tool in this picture upon which the more complex and intelligent collabora-
tive agents must be built.
The extend of collaboration in terms of AP goes beyond design, but it also involves social participation and collaboration. The users are and should be the primary target of architectural products. As a profession architecture is one of the most interactive and multidimensional ones. The architect is viewed as creating a work of art in some platforms, yet he never has and should have the freedom in terms of production. AP is one of the media where this gap could be bridged.

Direct public involvement in decision making processes cannot be avoided. This requires the provision of effective communication media in order to provide suitable grounds for public participation in design. With the take off of the internet as an effective communication media data support must be designed with this aspect of the profession in mind. The data needed for the generation of layouts is minimal constrained data which can be formulated without an application platform to handle AP. The purpose of such a tool should go beyond that goal and design a data support engine that is extendable to these communication and social interaction platforms along the design process.

3.2.3 Feasibility Studies

The amount of data that has to be processed towards the formation of a comprehensive AP is quite broad, scattered, multi-typed, and multi-resourced. A list of the data required for a typical project may include all available client data, functional program data, historical growth patterns, staff and employee projections, organization charts, growth projections, organization culture, current occupancy data, comparable building data, existing site improvements, space standards, personnel space standards, industry standards, economical and financial parameters, schedule and time constraints, site survey, government and regulatory agency approvals and schedule, utilities service, climatological data, traffic analysis, statistical data, site amenities and restrictions. It is possible to expand this list much further.

The way we practice architecture today has changed quite a lot compared to the spontaneous, architectless vernacular days where time and economical resources were not as primary. Today, we want to be able to draw conclusions before we spend anything and formulate the end product accordingly. The above list provides just a cross section of the information needed to be processed towards this goal.

A computational tool claiming to aid in requirements modeling in AP could have various computational future uses in this respect as:

- aiding in functional design
- aiding in collaborative information exchange
- aiding in initial decision making in feasibility analysis

The comprehensibility of the data must cut across these various domains if it is to assist in the process of intelligent tool support and computability of design because otherwise development of such an application does not teach much and does not improve the current conditions of the practice.
3.2.4 Functional Design Decision Support
AP prescribes the process of data gathering. Usually the functional program and the budget constraints determine the first iteration and the starting scope of the overall process. The computational application of AP must allow multiple views of the data, like grouping the spaces in data sheets, multiple groupings, diagrammatic information matrices, organizational hierarchy trees. Such capabilities allow the organization of functional requirements of a design towards the creation of an actual architectural design towards realization.

3.2.5 Performance Analysis & Evaluation
AP defines the boundaries within which the physical solution can be evaluated. Environmental design evaluation has become one of the primary expectations of clients from the architectural building products. Design evaluation is concerned with assessing the effectiveness of designed environments for users, while staying in environmental, economical, energy and comfort standards. A computable AP tool should generate a product that performance analysis and product evaluation can be checked against.

3.2.6 Reverse Engineering
A common data support across different applications to serve along the design process should support bi-directional data exchange. In shared platforms this is a more plausible and easy to achieve goal than stand alone systems. The ability of the application domain to extract requirements information from realized designs gives the system the ability to increase the capability of knowledge-base, re-use successful designs, learn from mistakes, generate a seamless communication in terms of iteration throughout the design process and integrate the AP to the iterative nature of problem solving the domain of design.

3.2.7 Knowledge-based Design
AP has been seen as a valuable resource for a systematized process that provides a structured framework for accumulating and classifying data [Sanoff 92]. It is agreed upon that AP is the organized collection of specific information that involves developing, managing, and communicating; therefore the management of this information after it has been created and used also must be discussed.

The requirements that design products have to fulfill are usually as the design process ill-defined and ill-structured. Not all the requirements are specified and the designer has to make assumptions. An ill structured domain like design implies that the problems in such a domain are unique implying that no classes of problems exist in the sense that solution principles can be developed to fulfill all the members of the class. In this respect, AP is an applicable domain for knowledge-based design process.

3.2.8 Multiple System Integration
A building as an end product of the process of design is a the totality of multi-systems that function together in terms of the initial requirements and criteria set, hence AP. In this scope the AP produced as an extend of requirements modeling is not only limited to the data that is required to generate functional layouts. The performance analysis, mechanical, electrical, and structural systems and their impacts all depend on the
requirements produced. The current research in CAD is headed towards producing tools which are capable of integrating behavior with layout generation in order to facilitate the multiple-system integration of the building design domain [Suter 99].

Along the same line of thought the dynamic requirements modeling is another domain of integration of the multi-disciplinary nature of building design and physics. It is a platform where information exchange, cross checking and evaluation across different sub-domains for constraint, behavior and standards checking could take place.

### 3.3 Scenarios from Practice

Scenarios give an idea about how the application tool should assist the process.

#### 3.3.1 Sanoff’s Scenario [1992]

1. Programmer devised standardized data collection sheet.
2. Department managers, during interviews completed data sheet, cataloging activities, personnel, space, and equipment and storage needs.
3. Programmer compared secondary activities (those necessary to accomplish primary activities) with each other in a relationship diagram for each division or department.
4. Activities are grouped from matrix into correlation diagrams of principal functions within each division or department.
5. Programmer prepares department/division/function interaction matrix for use in interviews with department managers.
6. Department managers identified relationships by matrix among functions/divisions/departments.
7. Programmer mapped relationship patterns from matrix in an interaction net.
8. Programmer administers “perception survey” to obtain user evaluations of office environments.
10. Programmer documents findings.

#### 3.3.2 Kumlin’s Scenario [1995]

1. Create a program team, there are three possible alternate scenarios in how to go about this:
   - Have the project architect take the responsibility
   - Have an in-house committee do the programming
   - Hire a consultant

2. Have an initial strategy meeting. The agenda should include determination of schedule, resources and logistics and identification of all data available.
3. Global decision making meeting. The purpose includes the description of the purpose of the project, review of the assumptions the program will be based on (e.g. overall cost, occupancy, site characteristics, schedule). The decisions made will include:
   - Image foreseen
Rationale of AP during Early Phases of Design

- Materials, relationship to environment, initial massing desired if any
- Working environment, natural resource usage, mechanical systems, interior space usage (e.g. open office versus closed)
- Employee amenities, eg. parking, food courts, lounges
- Visitor amenities
- Organization of program elements
- Change and growth

4. Gather data. A space list format that includes columns for personnel tabulations and projections for both space and occupancy, an adjacency matrix or similar format to describe both interdepartmental relationships and interdepartmental relationship spaces, a relationship diagram to illustrate graphically the departmental organization, the relationship of groups and parts, equipment data sheets,

5. Evaluate data.
6. Interview users

3.3.3 SEED Scenarios

In the context of SEED modular interaction scenarios were studied by Cumming. The scenarios that are defined by Cumming, although not explicitly mentioned, hint at the important role of data exchange in collaborative design environment through multi agent tools like SEED modules.

Incorporating the data support engine to Cumming’s scenarios we can expand the collaborative interaction.

a. Linear usage of SP, then SL

b. Linear usage of SL, then SP

c. Iterative usage of SP, then SL, then SP again and finally SL

d. One person is assumed to perform all design activities prescribed in SP, yet in SL two architects design in two separate concurrent sessions.
These scenarios assume one person using SP. However, as the professional scenarios taken from AP domain clearly indicate the nature of requirements generation is teamwork and collaborative. Cumming also includes in his study scenarios which involve meetings between the client and the architect and other involved expert parties where the above scenarios become middle way [Cumming 99].

The above scenarios due to their aim of incorporating the modular nature of the design process defined in SEED leave SP as a tool in a more documentary support rather than evaluating and collaborating. Even in cases of conflict resolution meetings the case of data sharing is not incorporated into the scenario. Once a consensus at some level is reached the SP tool support can become concurrent rather than parallel. Therefore, in this context the below collaborative scenario becomes important to consider with data support.

e. Multiple users perform activities in SP, followed by any of the above scenarios.

3.3.4 Evaluation
A computational tool to support the AP process can be aimed to serve various stages of the scenarios prescribed above. The current tool support devised in SPII, especially in relation to the data support expected from a shared database, is utilized only for the last stage, documentation.

3.4 Conclusion
Every integration problem bears two sub problems in its nature, one the requirements of the total system architecture that it has to be a part of, other the requirements of the domain it is supposed to serve. The nature of the AP as requirements modeling and the professional scenarios have been outlined to describe the scope of the domain of the problem. The requirements of the application domain reside in the predefined requirements of the SEED system architecture. The decision support with dynamic representation of multiple system requirement models in building design should aim to satisfy both sets of requirements in the application domain.
Rationale of AP during Early Phases of Design
4. Information Exchange & Building Product Models

This section will provide a background on information exchange models as the basis data and decision support tools depend upon. The application domain of SEED in general and SPII specifically benefits from various data model views of the past couple of decades. The background study is intended to bring to attention the relevant comparative models in information modeling in the domain of architecture.

4.1 Introduction

In the context of SP II information modeling takes place in two levels.

1. SPII being one of the modules of the SEED project has to abide by the information exchange through the shared schema,

2. SPII using a product modeling abstraction to requirements modeling in AEC as an extend of AP should be able to manage multiple product specifications through the shared schema concept employed in the overall project

Such a two level modeling necessitates the understanding of background in different product modeling and standardization efforts brought in AEC domains.

4.2 Information Modeling in CAD Tools

Computers, although mostly in the area of drafting and geometric modeling, has been a part of design technology for about three decades. The information modeling and exchange part of the problem is relatively newer. An information model for a product spans the products life cycle from design, fabrication, operation to maintenance, [Eastman & Fereshetian 94]. As the collaborative and collective nature of the design process become more understood the data exchange and information modeling aspects have been addressed more. The hardship of most information and data models in CAD mostly lie in the hardship of agreeing upon one data model to represent buildings and in universalizing such models.

Interest in software tool development to address different levels along the design process beyond drafting and geometric modeling, like problem specification, generative design and design evaluation has been the motivation initiating interest in information modeling in architecture. The common understanding that in one way or another design-
ers refer back to precedent designs caused research on design knowledge-bases and artificial intelligence paradigms to be a part of the research in CAD.

During the past two decades CAD researches have shown interest in information modeling of the domain of design towards improving the process and the products. Efforts in both specific and general standardized approaches have been pioneer in trying to carry CAD tools into a supportive role rather that the imitative one that they have been playing. This section will give a background on the most successful ones of these

4.2.1 AEC Industry

The AEC industry has many diverse disciplines such as architecture, structural and C engineering, cost estimation, construction, facilities management, electrical engineering, etc. Each discipline has its own distinct tools, terminology and processes while the challenge is the communication and coordination of these diverse disciplines around the same goal. Often the realization of a building project from design to its construction and facilities management is dispersed and become the central problem of the industry.

The information age has brought the disseminated business world where companies undertake projects in international platforms and outsource without distance boundaries. Abstract information modeling and computer based information management are the challenges for the tools to be developed.

The key requirements of AEC information models extending after Zamian & Pittman [1999] are

1. Comprehensiveness
2. Supporting multiple views, text, graphics, video, and sound
3. Non-redundancy
4. Extensibility
5. Abstraction, means to hide unnecessary details
6. Classification
7. Inheritance
8. Generation
9. Component based abstraction
10. Multiple referencing
11. Multiple functional views
12. Extensibility to new functional views
13. Data cumulation and evolution
14. Linkage to external libraries
15. Industry code and standard foundation links
16. User defined extensions

SP II in the context of SEED is the module where the AEC information is captured as a shared platform of requirement specification, therefore the data support architecture should keep such requirements in its priority.
4.2.2 Data Modeling Perspective

Underlying the structure of a database is a data model, a collection of conceptual tools for describing the real-world entities to be modeled in the database. A data model provides the basic tools for describing the data, relationships, constraints etc. of the information which is stored in a database.

Data models differ in respect to the semantic details they can express. Using a specific data model conceptual models can be built. In a conceptual model only the information itself is modeled, not the format in which the information is stored. This in turn is specified in physical data models.

Whether it is possible to find a formalism to expressing a product model standard which in some way is compatible with as many different data models as possible has been the challenge. Many research projects have favored the approach of choosing a single model. In the context of SEED project this approach have been pursued in the AP domain in the first prototype developed, SP-I [Donia 98]. The disadvantages of such an approach have been seen in:

- changing the set of design specifications in terms of adding new types of specifications
- selecting subsets of specifications with which to work
- supporting a formal way of defining complexity of modeling of design requirements
- combining both prescriptive, in terms of properties of materials and performance style requirements modeling, resulting in inadequate modeling.

The functionalities that data models of design applications should have are:

- coexistence of objects in different stages of completion (complete and incomplete designs)
- distinction between representation and interpretation of data
- ability to ask and evaluate hypothetical queries
- encapsulation of non-determinism and a notion of choice in the data model

Especially for data models to support intelligent agents they must be able to handle incompleteness and non-determinism during the early phases of design [Imielinski et al 91]. A predetermined data model to represent the final design is conceptually does not represent the idea of neither design nor requirements modeling stage. Most of the requirements and specifications expected from the end product in building design might overlap, however as various sets and types of changes occur these requirements also start to evolve and change form. Not always the same descriptor can capture the change propagation. Multiple modeling capabilities in design applications are a logical way to pursue, yet the looseness of data and information to be processed through the data must be communicated seamlessly and semantically correctly to the end user. The end user should not be expected to know and understand the data model and schema representation that a database application requires in terms of implementation.

Users of such data models need to be introduced to a system where they can understand and use the data from their perspective. Architects when designing do not necessarily think about the requirements details as attribute lists, but they need to abstract that information in terms of spaces, the relations of spaces to each other and in most cases they...
need visual references rather than textual or spread sheet data. [Ekholm et al 2000].
Moreover, a product modeling abstraction should employ different views, construction method view and the data associated with it could be differently expressed than the spatial view [Eastman & Siabris 95, Bjork 91]. The different views that a product model should employ towards design support can be enumerated to others as well like structural, cost-base, performance-based, etc.

In terms of data modeling for architectural information presentation there has been many examples in the past decade especially. The standardization efforts followed by STEP has increased the interest. There are many models developed and is being worked on which aim to capture the requirements in building design as correctly and inclusively as possible. Conceptualizing space for building classification and design [Ekholm & Fridqvist 00], integrated building models [Augenbroe 91] are some examples.

The choice of data model for expressing a product model standard is a fundamental issue which has many consequences on the development of software utilizing product model data. Typical features of engineering design process in representation of product models can be in different levels of abstraction. The process is not linear and specification and generalizations must take place simultaneously.

A solution to the problems in the building and construction industry is seen as an open (vendor independent) approach that allows each participant in a project to use the application systems of their choice for building and maintaining a project database. Requirements of such a system are:

- it must support the sharing, storing and exchange of product models and projects
- experts, such as structural and HVAC engineers should be able to employ their preferred systems utilizing the shared project data and extend the product models and projects to including their domain specific information [Tolman 99].

The concept applied in SPII extending AP into product modeling could be extended to meet such requirements that AEC industries impose.

4.2.3 Universe of Discourse

The objective in all data and information modeling is to describe a universe of discourse (UoD), the subset of the world to be modeled that is of interest to a specific information system. Thus developing an information system involves a formal description of an abstract model of a piece of reality (the UoD) [Loucopoulous 92]. The scenario assumed in SEED is that "agents can enter or leave the collaboration at any time; they are not restricted to a specific UoD or data model, nor a specific programming paradigm" [Snyder & Flemming 99]. The same is true for SPII in terms of actors, they are presented with concepts of product modeling, the real world to be captured is not bounded by any UoD.

4.2.4 Database Support

A database is normally used to maintain a model of some aspect of reality. Traditionally a database is a repository of information about individual objects.
The state of the computer hardware technology has usually been the primary influence on the development. In 1960s and early 1970s typical applications involved the manipulation of large amounts of homogeneous data by storing it as records and the physical format was as compressed as possible. In 1970s the basic theory of databases evolved and in 1980s the commercial relation databases have taken their permanent place like word processors on our desktops. The integration of the support into architectural, engineering and construction along the design process has been an area of research since 1980s.

The traditional data models which are relational, object oriented and object relational will not be discussed in detail in. Although they form the core of the formation of complex data models to represent real world situations their historical evolution and capabilities are not relevant to my concentration. Database support is such a tool which should be utilized as comprehensively as possible to build intelligent CAD support in architecture is the core aim.

4.3 Similar Research

Data modeling as a core supportive functionality to intelligent CAD tool support has been the motivation of numerous projects. Although it is very hard to cover all of them in detail some must be mentioned in comparison to SEED system architecture and the product modeling concept pursued in SP-II. SEED as its core software architecture idea does not employ a data modeling as a building product abstraction perspective. [Flemming et al 95]. The independence of the different modules to be developed along the life cycle of the project is what gave rise to the concept of product modeling as an extend of AP [Donia 98].

4.3.1 Shared Data Support

Correct information exchange is achieved through being able to communicate the semantic concept underlying the data.

Semantic integrity necessitates the satisfaction of criteria beyond correctness like:

- robustness of information exchange
- approach to deal with inevitable information loss along during exchange
- semantically correct conceptual bi-directional mapping between different schema
- synthesis and re-synthesis of concepts
- variations between static and dynamic information

Among the many projects which address similar issues in data integration there are quite a number which address issues of shared data exchange and information support towards integrated decision making [Fenves et. al. 94, Wong and Sriram 94, Khedro 95, Flemming et. al. 96, Wittenoom 99]. Some of these efforts must be summarized for their comparative nature to SEED in general and SP-II specifically.

ICADS

The Intelligent Computer Aided Design System (ICADS) was a research project in cal Poly, San Luis Obispo in the CAD Research Unit. Specifically the project uses a relational database implementation of a design prototype knowledge-base of prototypical
building types and site descriptions. Instantiations of a building prototype and a site description are referred to as the design program (equivalent of AP concept in SPII).

For the ICADS system to be used the design program has to be established first. It is done manually except the results are stored in the database, thus a linear approach is pursued. The ICADS system has intelligent design tools (IDT), i.e. expert systems of access, thermal control, cost, lighting, acoustics and structure which are Control IDTs. The expert systems are given the information from the design program, the database is queried and the resulting information is translated into the expert system’s representation. The geometric interpreter (GI) than pattern matches a predefined design program against the objects of the program to fill in space attributes. Once the spaces are created the expert systems execute evaluations against the information found in the design program [Myers et. al. 92].

The product model representation of the design program which served as a source for the features of ICADS was criticized for being error prone, having severe performance and maintenance problems and lacked notion of class description. [Snyder 98].

FIGURE 4.2. ICADS System Software Architecture: [Myers et. al 92]

**EDM-2**

The Engineering Data Model (EDM), the predecessor of EDM-2, is a modeling environment developed by Eastman, but it was not based on object-oriented paradigms [Eastman & Siabris 95]. EDM-2 was developed to incorporate object structuring techniques in terms of specialization lattice and composition hierarchy into a product modeling and database language to support model evolution. The types of evolution supported as described by the authors are

- translation between distinct models
- deriving views from a central model
- modification of an existing model
• model evolution based on writable views associated with each application [Eastman & Jeng 99]

EDM-2 assumes that no one building model will be able to bring together all the data requirements of future intelligent design applications. It is proposed that multiple views of product models which are extendable and which can be evolved into other models will be needed.

Model evolution is based on mapping between models. External maps are those between the application and the application view, providing an isomorphic mapping between it and the application. An Application View is equivalent to the file formats of the application for storing project data between sessions. Internal maps are those between Application Views and the building model, carrying all logic of class conversions and other changes required in data translation. Such mapping are provided assuming that different applications would have their own view of the data and to manage data redundancies between the central building model and other application views. [Eastman & Jeng 99]. EDM-2 can be viewed as a database schema definition language with constraint definition and object management capabilities.

FIGURE 4.3. EDM-2 General overview of model evolution architecture: [Eastman & Jengs 99]

EDM is seen to be in closest spirit to the information exchange model developed in SEED [Snyder & Flemming 99].

SEMPER

In the domain of building design generation and evaluation there are successful projects that employ shared static object models as their core data model and develop intelligent agents around this shared model. An example to these is the SEMPER project developed at the Department of Architecture at Carnegie Mellon University.

The dynamic links between applications occur at the object model level though derived values, avoiding direct links between application data structures. The applications developed communicate with each other though this agreed upon shared object model, yet still remain fairly independent in implementation level. [Mahdavi et. al. 99]
4.3.2 Standardization Efforts

The efforts of developing semantic standards for the building and construction industry started around 1986 with the STEP AEC group.

The idea of developing general systems models to capture multiple levels of requirements for different sub-expertise domains in AEC industry came around the same time. The general AEC reference model proposed by Geilingh (GARM) was influential for quite some time.

**GARM**

GARM allowed the representation of rather complex subjects in a seemingly simple model [Figure 4.4].

GARM provided a distinction between “requirements” called functional unit (FU) and “solutions” called technical solution (TS).¹ FU's have requirements while TSs have characteristics. An FU can be satisfied by one or more TSs. A TS decomposes into one or more FUs of a lower order. GARM could not succeed although it was seen as a product modeling language in which generic product type models for specific classes of products like roads, channels, buildings can be modeled. Such a language was thought to be useful only if more or less complete set of product type models were available [Tolman 99].

![GARM product model representation decomposition: [Tolman 99]](image)

**STEP**

The Standard for Exchange of Product Model Data (STEP ISO 10303) is an ISO committee with the purpose of developing an interdisciplinary generic product information and a data exchange model. ISO 10303 was accepted in 1994 and it is continuously being extended. The underlying intention is to define a uniform representation of prod-

1. The FU definition in GARM differs from that of SEED. In SEED a functional unit (FU) is defined as representing a combination of functions to be satisfied by a single design unit and also serves as a repository for requirements of that design unit, a spatal or physical part of a building. An FU in SEED can contain other FUs. [Akin et al 95]. This is a very important example of the problem of semantic integrity in the realm of AEC in early phases of design towards generalized software tool support.
uct information and to provide mechanisms that enable the exchange of product data between different computer systems over the complete product life cycle. [Mannisto et. al. 98]. Data models in STEP use EXPRESS and EXPRESS-G, having graphical notation, language which itself is also part of the standard.

The fundamental structure of STEP prohibits modeling of products in an object oriented manner, which is seen as one of its major drawbacks apart from the organizational bureaucratic idiosyncrasies which prohibit STEP to evolve as a widely accepted standard in either building or other industries. In addition, the lack of feedback to clarify STEP concepts and how they should be interpreted also is seen as a reason for the fast elimination of the approach.

STEP employs some general concepts to product modeling through EXPRESS. A generic product structure supports basic concept of a component having other components as parts. In addition, specification of classification types is also supported, allowing sharing of information to the sub classes through inheritance. STEP aims to define a schema that allows each specific industry domain to represent its own product data as instances of the standardized schema.

SP II in this respect tries to follow the generalized approaches, like those in STEP, to product modeling, hence AP as requirements view of building products, and introduces a software tool for the purpose.

4.3.3 Information Modeling Languages

The requirements of large scale information management systems lead to the evolution of high level modeling languages to model functional application requirements and information system components at a conceptual level. Information modeling languages, unlike traditional programming languages, provide capabilities suited to building and managing information in a complex, computable model and have capabilities that in programming languages require manual programming [Snyder et. al 95].

Several features that further describe the power of information modeling languages after Snyder & Flemming [1999] are:

- semantic object relationships which in object oriented programming languages must be manually established. In some modeling languages object compositions using relationships between objects
- modeling languages allow specification of reactions to events that occur on an object, like an event triggering an action or constraints on structural or data properties of objects.
- they allow dynamic and temporal capabilities like version and configuration management
- unlike traditional product models where process and product models had to be represented independent of each other modeling languages support homogeneous representation of both.

An information modeling language should be implementation independent and support existing and future components without having to re-engineer or alter code. The shared model environment that is designated by the modeling language is the model that
applications in the integration network should abide by. The internal representations of each application must be mapped against the shared schema only. This does not necessitate definition of all the internal representation, only those that are required by the agreed upon shared environment could be mapped.

EDM and STEP/EXPRESS efforts can be viewed as exemplars of high level information modeling environments. SEED project also takes such an approach as its information exchange model [Flemming & Woodbury 95].

4.4 Conclusion

Although all the studies sited in this section address the issues of a common knowledge space to manage the requirements of the AEC industry, they agree that no one view can be preferred over an other to provide seamless decision and computational support for the profession none of them bring a fluent and flexible environment proposal. The approaches carry characteristics of extendable views, like SPROUT, different view generation, like EDM, shared object instantiation and agent specific extensibility, like SEMPER, yet they do not address generality. The generality assumed in STEP efforts have proven to be unsuccessful for bureaucratic reasons and the models developed have not been accepted by AEC industry although some industries do employ these successfully, like automotive. Therefore, research of tool support for the domain of dynamic requirements modeling should learn from the problems posed in the above described studies.
5. Application Domain

This section is devoted to describing the modular architecture of SEED in respect to information sharing and exchange, and exemplify how it is extended by using SP-II. The potential problems and challenges will be underlined along with problem design using specific implementation issues drawn from SP-II. Seed Layout will be used as a comparative example module when and if necessary.

It is important to note that this section only aims to study the current existing implementation towards solving the integration problem described. The suggestions brought will be inspired and limited to the current system architecture. The aim is not to counter part what already exists, but it is merely to try to design a prototype and bring to attention mismatches to serve future research that could take place in a similar contextual and application domain.

5.1 Current System Architecture in SEED

SEED contains a collection of modules that come around a set of supportive tools. [Figure 2.1]. Although the development and internal representation of these modules are heterogeneous, they are intended to appear to the user as a part of a unified whole, therefore the development of the modules are based on a common logic shown in Figure 5.1.
Each module uses an independent representation that is particularly suited to the tasks addressed in the module, thus information exchange requires translation between representations based on shared concepts. Currently SEED project is composed of three modules, Seed Pro, Seed Layout and Seed Config.

There are three central conceptual constructs that the modules have [Akin et. al. 95]:

- **Design Units (DU)** are the basic spatial or physical entities that make up the representation of a design in Seed Layout and Seed Config, e.g. a room or a wall
- **Functional Units (FU)** collect all of the requirements that the design unit has to satisfy in a single construct, e.g. a wall may have to satisfy certain sound insulation, thermal resistance, visual privacy and load bearing capabilities, a functional unit carries all of these requirements of a wall
- **Specification Units (SU)** collect the design intentions and criteria to be satisfied by a functional unit

The representations that are generated by each module are complex, information rich and object based. Therefore, database support is inevitable. The teams developing individual modules are expected to rely on the same database system, interface builder, or geometric modeler developed independent from SEED [Flemming & Woodbury 95].

### 5.1.1 The Architectural Programming Module

SP-II accepts as input a collection of specification units that capture the general goals and requirements of a project. One of its outputs is one or several functional unit hierarchies that form an architectural spatial program in the overall scenario of SEED. However, the intend is to be able to support other future modules which might benefit from such a set of specifications as well, like a cost analysis or structural analysis module. The outputs of SP-II are solutions which become inputs for these intended modules.

Design specification are defined in SP-II as a collection of design intentions and criteria to be maintained by the architectural program. The construct that SP-II depends on is that a general requirements modeling approach serves the profession better.

**FIGURE 5.6.** SP-I’s project structure: [Donia 98]
The first prototype SP-I followed a static approach to requirements modeling. The specifications in this model were predefined into five categories as building, site, budget implementation schedule and client profile, each of which were left to be further detailed.

The object model of the first prototype [Figure 5.6.] designates certain sets of pre-defined attributes for the architectural programmer to fill in to generate plausible output hierarchies of FUs. The user has to fulfill the data set prescribed by the object model of Figure 5.7. for the building category. However, this model limits the scope of the architectural programming module to other SEED modules only. The development of the second prototype was in the spirit of generalizing the domain to a broader community in architecture, thus a product modeling abstraction and dynamic programming approach was pursued [Donia 98].

The comparison of the two prototypes is crucial in understanding the general system architecture and information exchange foreseen in SEED in reference to any intelligent agent which is to benefit from such a data set.

The objectives of the second prototype is described by its author as:

1. to provide the degree of flexibility needed to create and manipulate design requirements, define product models and generation mechanisms and structure generated outputs independently from the design requirements description
2. to be able to perform all these tasks without the need to re-program and re-compile the system
3. to provide flexible constructs of defining design requirements
4. to support an open ended architecture where new models of specification can be generated
5. to support associations between different models of specification to generate system specific outputs
These objectives in an integration task have to be further combined with the general objectives of SEED as sited in section 2.2.

As a part of the integration problem, the discussion of the shared schema decisions and the system architecture employed is not in the scope of this work. On the other hand, as an information exchange problem the study of each module’s system architecture has to be decomposed. As a result, functionality problems, usability issues and conceptual mismatches become a part of the problem to be solved.  

**FIGURE 5.8.** SP-II’s product modeling object model: [Donia 98]

SP-II is composed of objects to support definition of product models, creation of generation mechanisms, generation and manipulation of design requirements. The generation mechanism provides rules for creating customized outputs, i.e. solutions, from a design requirements description. The generation of FU hierarchies, or SU models, takes place at this stage. Hence data retrieval and saving features also are a part of this stage because the output model is the model that should get stored in the database according to the initial scenarios developed.

In this framework SP-II should be able to store in the database not only system customized outputs, e.g. an FU hierarchy as an input to Seed Layout, but also must be able to define an SU and product modeling database. Storing an SU and its constituents in the database necessitates the creation of each product model as a reference frame. The type of SPClassifier and SpecCategory objects a SpecUnit can refer to and contain is determined by its ProductModel object that it is related to, as can be followed from the object model in Figure 5.8.

1. A usability study, to which I have participated as an evaluator, was conducted for the current version of SP-II, focusing on the project and programmer use cases. More discussion on the findings of the study, which mainly concentrate on HCI perspective can be found in Erhan 00.
The **ProductModel** has one to many relationship to **ClassifierGroup**, **SpecCategory** and **RelationType** objects. This allows the dynamic generation of requirements modeling according to the user needs and become the attributes of the product model to be generated.

### 5.1.2 Information Exchange through Shared Schema in SEED - SPROUT

In the SEED context the common universe to facilitate inter modular communication [this also in its nature includes the communication of each module between project sessions in each module] is established through a special modeling language, SPROUT [SEED representation of **Processes**, **Rules**, and **Objects Utilizing Technologies**] [Snyder et. al. 95].

The motivation behind developing SPROUT as a middleware to support the database backup required by SEED modules was the lack of a robust database system which supports database management, case-base management, as well as version and configuration management [Snyder 98].

Each module in SEED has to define the objects and attributes that is required for the module to store in the database in terms of SPROUT. A brief of definition of terms used in SPROUT is beneficiary at this point. The below descriptions are limited to only those which are currently functioning in SPROUT.²

**Schemas:** A schema describes the structure of a collection of information composed of **domains**, **relationship types**, and **classes**.

**Domains:** Domains are data-type specifications and establish a legitimate set of values in the same way as database domains.

---

² For a more detailed description on SPROUT refer to [Snyder et. al. 95], [Snyder 98] and [Snyder & Flemming 99].
**Application Domain**

*Relationship Types:* They establish the behavior for a relationship. Minimally they have a name and a container (set, vector, bag or link). They establish certain kinds of associations between objects, e.g. one-to-many.

*Classes:* A class is a collection of attributes, some of which may be inherited from a parent class. A class can be viewed as the complement of a class in object-relational database terms.

*Values:* They are name/domain pairs and are complement of attribute specifications.

*Relationships:* They are attributes which specify “links” between objects and imply a containment relationship between two objects, e.g. an SU may have many SUs. The containment types are defined via the relationship type and they specify the class of objects allowed in the contained.

---

**FIGURE 5.10.** General Information Exchange System Architecture in SEED

One of the most important contributions of SPROUT was foreseen to be the ability to automatically generate language bindings necessary to link a new application to the system. Construction of complex models do require the assistance of the computer. Code generation capabilities would provide a significant automated help for such multi agent software architectures. The automated mapping generation capabilities were not implemented as anticipated. Therefore the language bindings and database mapping code to link each agent to the overall system for any type of information exchange has to be manually provided.  

Linking SP-II to this architecture involves managing certain software design problems due to the internal representation of the AP agent. These are:

1. To be able to establish communication with SP-II through the shared schema the shared schema has to be extended to cover SP-II objects. As described in the overall SEED architecture section the basic AP information object is defined to be an SU.

---

3. The first prototype implementation of combining the communication modules, the database connection and the individual agents is still under development using Seed Layout and involve my active participation.
The first prototype SP-I supports a static SU definition. However, in the second prototype the attribute and object composition of an SU changes with each product model and generation mechanism, for solutions are dynamically specified. There are two possible paths that emerge:

- using the first prototype since it is in the spirit of the shared extendable information exchange in the predefined universe in SEED.
- tackling the problem from the point of view of the dynamic programming perspective and designing a prototype sub problem which will help identify the pitfalls and open new paths.

The first approach ignores the findings of the group which lead them to the development of dynamically managed functionalities. A dynamic framework is also more promising for the domain of decision support in architectural product development since it addresses collaborative, asynchronous and synchronous, heterogeneous, and distributed nature of AEC requirements management. Therefore, the second approach is found to be a more plausible research methodology.

2. SPROUT schema extensions that SP-II requires can be managed by writing a rule based schema generation engine. It would involve recompilation of SPROUT each time and reloading of the database schema, which break the seamless computational support from the user’s perspective. This means with each new solution generation in SP-II a new SPROUT extension to the shared schema, hence an extension to the database schema, will evolve. Since the automation of the language bindings do not work, for each new model a new mapping code would have to be written or would have to be revisited based on the semantics of the new products that are needed to be saved in the database. Or a general mapping mechanism has to be developed which requires the anticipation of possible output generation mechanisms that could be used. In both cases the design problem exceeds the maintenance work of database connection because if any additional capabilities is required it would mean re-visitng of the implementation details of quite a couple of modules. Thus, the design of information exchange in dynamic platforms should not be forced with static frameworks like shared schema, although schema evolution is supported. A more generic architecture has to be provided.

The sample problem design section will address all these issues with agent specific implementation details.

5.1.3 SEED Database

The database selected after the evaluation and comparison of a number of other object oriented database applications for the project is UniSQL [Snyder 95]. UniSQL is an object relational database which combines characteristics of object oriented and relational databases.

Ideally none of goals of individual components should be sacrificed in the process of integration. The goals of the SEED database can be summarized as:

- persistent storage of design data of interest,
- interactive browsing and retrieval of stored object,
- configuration management,
• inheritance, when an object is retrieved it must inherit all properties and behaviors of the superclasses to which it belongs,
• multiple classification, e.g. a private office must be retrieved both through offices and private spaces,
• prototypes with default specifications, and
• constraint management.

The current database architecture supports most of these through SPROUT. In the context of SP-II where it is necessary to create multiple schemata the usability of information modeling language becomes an overhead rather than supportive. Abiding by domain specific output model based information neglects the potentials of the central decision of SP-II, dynamic information generation.

5.1.4 Agent Communication Model

Given the requirements and design followed in SEED no practically viable communication framework that could support concept mapping between a native programming language and a data modeling language was found. Therefore, the implementation of one also became a part of the SEED software architecture. [Flemming et. al. 00]

Initially the agent and database communication was designed to be a direct one, however three disadvantages were seen in this approach:

1. The code targeting one database could not be used for another one,
2. Agents must incorporate communicating directly to the commercial database which requires translating between imposed API of the database, data structures of SPROUT and the internal representation of each agent,
3. Communication becomes a manually managed responsibility of the agents

To overcome these disadvantages different models were pursued all of which are described in detail in [Flemming et. al. 00]. The final system design which was found to be the most plausible one was separating the communication and database agents and connecting all modules to these two servers to facilitate communication [Figure 5.11.].

**FIGURE 5.11.** Agent Connection Architecture in SEED: [Flemming et. al. 00]

The DBServer provides persistent storage services via set of communication protocols, while the CommServer propagates design products as process events by using interest
registration. The CommServer is a client agent to the DBServer because the CommServer needs to retrieve design products from the DBServer for some of its operations.

The links between all these agents are established through mapping codes. Generality is the basic concern. The mapping code between UniSQL database and the DBServer is written against SPROUT, i.e. the constructs in SPROUT as defined in section 5.1.2. are used, not a specific schema defined through SPROUT. This enables new agents to enter the system without much computational overhead since as long as SPROUT is used the constructs are those provided by the modeling language.

![Diagram of SPROUT Compiler](image)

**FIGURE 5.12.** SPROUT Compiler: [Flemming et. al. 00]

Java is used as a data transport medium to support dynamically typed systems. Although Java is a statically type language, a Java Virtual Machine (JVM) is a dynamically typed system. Although in Java programming language class definitions cannot be loaded at run-time, the underlying JVM is capable of doing this, aiding hassle free data transport.

SPROUT compiler generates Java classes and a database schema according to input in the SPROUT shared schema [Figure 5.12.]. The .jar and .stream files contain class definitions to establish the client class context, i.e. Seed Layout or SP-II in SEED currently. A .jar file is a standard Java defined class file. A .stream file is equivalent to a .jar file except that it is for C++ instead of for Java. A .stream file is defined by the system. It is simply in Java serialization format. So, a .jar file is read by JVM, while a .stream file is read by a C++ program, like Seed Layout.

### 5.2 Evaluation

The generic framework of SP-II, following dynamic programming and product modeling approaches allow AP developers to define their universe more inclusively with their own domain specific attributes. This approach has major differences in comparison to a specific shared schema concept. Some of the concepts and current implementation decisions in the general information architecture of SEED pose drawbacks for pursuing the research objectives discussed earlier in the context of requirements modeling:

1. **The internal representation of SP-II depends on output model generation as solutions solely based on user specification.** There is no way to predetermine the SPROUT
schema to represent the output model. The motivation of SPROUT as a modeling language is not to support generality, although it supports schema extension.

2. Generating a SPROUT schema from an output model is a tedious, yet trivial problem ignoring the fact that there is not much to learn from the exercise. However, this approach brings with it issues that have not been addressed neither in the context of SEED information exchange nor in SP-II, like:

- **Semantic integrity** is left to the users of SP-II. It is assumed that their expertise level is able to cope with retrieving and evaluating the semantic correctness of the data. However, when multiple models are produced for the context of AEC, which is the shared domain that all the sited work in this study refer to, currently there is no feasible way to control an extended schema to deal with multiple levels of meanings. For example, a space can be defined with an area attribute or a volume attribute. Mechanisms to house both data interpretations require a semantic comparison SPROUT. And for the same matter a product modeling abstraction should not limit such interpretations. Chapter 3 and 4 has tried to bring to attention that the long on going research has at least managed to establish the common consensus that the interdisciplinary nature of the AEC industry is beyond meeting around common conceptual definitions and descriptions.

- Based on the initial common concepts of SEED, SUs are the central information packages that serve as CBR cases or information sets in SP-II. An SU is a specific instance of a product model. In the prototype library, an instance of an SP Model and an instance of a SL model are both SUs, but they do not have the same data model. In this context the product models become central to SU generation. A **CBR module** must be able to handle both SUs and requirement models.

- The role of the modeler in SP-II has always been seen as an expert who prepares the model. The programmer had the central role in instantiating a project of that model, which is merely a data fill in. This approach undermines the future research potentials inherit in requirements modeling and ignore output model generation potentials. The capabilities of output model generation through formula specifications between different product models increases the potential data intensity that has to be dealt with. Defining new products from already defined ones, inheriting features from precedents, defining part-subpart relations are all relevant problem solving and solution generation issues in AP when a requirements modeling perspective is taken. So far these issues were not central to the research, yet they should not be ignored in a decision support tool depending on persistent data storage.

3. Assuming the problem of extending the schema based on output models is solved, the current state of the information architecture still continue to become an overhead since the mapping code has to be either regenerated, revised or generalized. Generalizing, if possible, should be the plausible approach. It should follow the comprehensive user domain and intelligent agents to benefit from data generated by SP-II investigations, which do not exist yet. A comprehensive user domain study along with modeling intelligent agent communication scenarios would aid in dynamic data management. Such studies do not exist yet.

### 5.3 Prototype Problem Design

One of the aims of the SEED project was that AP and layout generation modules should be able to communicate in the FU level. Therefore, at least to demonstrate the imple-
mentation details and overhead for future references a preliminary prototype problem
design was seen beneficiary. Certain assumptions had to be made for such a prototype.

1. It is understood that although for the prototype to work the implementation has to be
completed in full. Most parts of this implementation will have to be repeated, revis-
it and in some parts rewritten for any other SU specification that SP-II would want
to reserve persistently in the database.

2. The sample model created, is only limited to the current shared schema definitions
which cover the objects that Seed Layout is interested in for simplicity. SP-II cannot
save SUs or any other objects that it is interested it unless the problem is redesigned
accordingly.

3. The interface issues that come up with integrating such a data management engine
are brought to the attention but are not designed since the scope of such a problem is
beyond a semesters work. Moreover, it has to include a study on intelligent agents
and how those should be integrated in the context of requirements modeling in AP,
which again was not the purpose of this preliminary work.

4. Initial assumptions were that the database support for the SP-II module would be in
SU level, similar to Seed Layout which is interested in storing FUs, Layouts, Layout
Problems, and Context. The studies presented above have shown that the level of
granularity in SP-II has to start from the product modeling abstraction. The classifi-
cation of APs as requirement data models into typologies and the knowledge-based
intelligent agents managing such classifications are based on the product modeling
information. How the user, in this context is AEC expert of any discipline including
the architect, is to be exposed to such information is an important HCI issue, which
should be designed and studied separately.

5. Since the implementation and testing of the integration of communication and data-
base servers was a task that took place simultaneously unexpected time problems
due to understanding and debugging had to become a part of the overheads of the
process.

5.3.1 Generating a communication model in SP-II

As an initial starting point in SP-II a plausible product model for the communication
schema between other SEED modules, in this case Seed Layout, has to be created. The
model has to include all the information that will be necessary for Seed Layout to start a
problem. Therefore, the modeling will take the shared schema as its guide. The model
attributes are based on these specifications. A new database test library is created with
example SeedPro (SP) and Seed Layout(SL) models [Figure 5.13.].

The action sequence in SP-II is based on output model generation from input models. In
its implementation the output models are also the same objects as the input models, i.e.
SUs. Such a generation mapping allows the sieving of unnecessary, redundant data for
the database or communication. The output model is also in its spirit a collection of SU
hierarchies. They are treated as FUs only because the information mapped is limited to
that a spatial functional unit would require. Other programming criteria like cost, struc-
ture, performance, interior architecture or materials are striped out.
The SP and SL Models created for the prototype have one-to-one generation relations to each other for simplicity purposes.

Notice that this model could have been organized in an other way with different specification, categorization and relationship type compositions, yet still carry the same information necessary for Seed Layout. In that case the mapping between the SPROUT schema and the SP-II objects would have to change accordingly.

In detail; in the above model the Context specification group is a part of SpecCategory class, which in the SPROUT domain is to be mapped onto Context class. Similarly the
LayoutProblem specification category should be mapped onto LayoutProblem class in the shared schema. LayoutProblem has relationship attributes SpatialFu and Context (so they are of type vector, link, bag or set), which can easily be defined as SpecCategory in SP-II, thus the mapping mechanism must further be able to detect such semantic interpretations and be able to define the one-to-one mappings correctly. This does not only involve the understanding of implementation details of all the agents in the communication namely, DBServer, CommServer, commercial database UniSQL, agents to communicate (Seed Layout and SP-II), the SPROUT modeling language and environment, but it also necessitates a thorough understanding of how to manage requirements modeling criteria in such multi system based domains as AEC. These issues must be addressed in designing a generic communication architecture for SP-II to provide knowledge-based decision support.

5.4 Design

The above example demonstrates the complex integration problem that arises. Yet, putting these aside the implementation design details, assuming that our UoD is limited to the above highly simplified AP domain, are further developed below.

Integrating SP-II into the overall communication framework involves possible extension of the shared schema providing necessary modules to link to communication servers, and creation of language bindings on SP-II side. The mappings between SP-II’s internal concepts and corresponding concepts in the shared schema are packed in the Java objects generated by the SPROUT compiler. SP-II also should be extended with the HCI components to provide communication connection and database browse. Since the language binding compiler and code was not implemented the mapping code has to be hand written. In the context of SP-II this has quite a lot of disadvantages, since the above exercise has to be repeated for every generative scenario if the current schema decisions and implementation details based on these decisions are not generalized. Figure 5.15. describes the components to be provided for such an integration.

![Figure 5.15. Implementation modules for integration](image-url)
In creating the mapping mechanisms some OO design patterns are extensively made use of like factory and handler methods. These methods will be described along with their applicability to the dynamic object declaration environment in SP-II in comparison to that of the static implementation pursued in Seed Layout. The bindings and mappings are aimed to match the internal representations of agents with those of the SPROUT objects.

5.4.1 Factories

The intent of the factory method is to define an interface (a generic class) for creating an object, but let the subclasses decide which class to instantiate. As a result, instantiations are deferred to subclasses.

The factory method takes care of when to create which object rather than implicit specifications, like when to create an SU or FU, in managing objects defined through modeling languages like SPROUT with an agent communication architecture. Such a pattern allows the addition of new classes to the schema with only taking care of the overloaded implementation of the new class methods.

Factory pattern is applicable when:

- a class cannot anticipate the class of objects it must create
- a class wants its subclasses to specify the objects it wants to create

For example when a FunctionalUnit object does not know whether the object to be created is a Building Fu or a Spatial Fu the factories take care of which method to instantiate. The naming convention of factory is used because as a result an object is manufactured.

The factory method is quite an applicable methodology since the essence of the communication architecture is to create various equivalents of the SPROUT objects and carry them across the database and agent modules.

SP-II Factory

The design of a SeedProFactory has to consider the persistent specific and static objects defined by the SPROUT schema since the initial prototyping decision was to stay in the limits of the already existing SPROUT schema for communicable objects. If compared with SeedLayoutFactory it is seen that SeedLayoutFactory makes use of specific attributes
and objects that are defined through the SPROUT schema, hence the database schema, like maxWidth and minWidth attributes of type real.

The correspondence of such attributes in the SP-II implementation, recalling back the dynamic modeling and programming platform, is a SpecPrimitive object which is categorized by a SpecCategory. SpecPrimitives are only defined as basic types which are integer, floating point, string, boolean and enumerated which is a set of string values. Thus, if maxWidth was not one of these primitive types of the modeling environment in SP-II it would not be possible to model it as a SpecPrimitive, hence the factory specifics would change.

Figure 5.17 shows a possible design for a generic factory that takes care of SP-II objects. The SPROUT schema could be matched onto a specific model through the implementation of a hard coded factory and set of handlers in SP-II, which manage current SPROUT objects of Seed Layout, like context, fu, layout, etc. To map SeedLayout objects to those of the information model through SP-II just requires the recoding of all SeedLayout information modules using SP-II objects. This approach assumes that the naming conventions and modeling specifications in SP-II will be maintained. On the other hand, introducing these specific objects through SP-II requires defining the generic SP-II modeling objects in any case. Therefore, the factories (and handlers) do not assume any predefined model and is a prototype design to handle existing generic SP-II objects in an information exchange scenario.

FIGURE 5.17. SeedProFactory, a generic approach

---

```
include Object.h OrdColl.h ObjArray.h JavaValueFactory.h SproutObjectFactory.h SeedProFactoryMacros.h

global #gSeedProFactory:SeedProFactory *
#projectName:char
#productName:char
#project_description:char
#mapObjects:OrdCollection
#products:OrdCollection
#libraryname:char
#inputSUs:OrdCollection
#outputSUs:OrdCollection
#productModelDesp:char
#categories:OrdCollection
#classifierGrps:OrdCollection
#relationtypes:OrdCollection
#relationdescription:char
#classifierdecription:char

+SeedProFactory=(InitInstance();)
+-$InitInstance(void)
+InitializeSeedProFactory():bool
+MakeSpecUnit():JavaObjRef *
+MakeClassifierGroup():JavaObjRef *
+MakeSpClassifier():JavaObjRef *
+MakeSpecCategory():JavaObjRef *
+MakeSpecElement():JavaObjRef *
+MakeSpecPrimitive():JavaObjRef *
+MakeRelationType():JavaObjRef *
+MakeProduct():JavaObjRef *
+MakeProductModel():JavaObjRef *
+MakeGenMechanism():JavaObjRef *
```
5.4.2 Handlers

In handler pattern the request gets passed along a chain of objects until one of them handles it. The senders and receivers are decoupled by giving multiple objects a chance to handle a request [Figure 5.18.].

Handler pattern is applicable when:

- more than one object handles a request, and the handler isn’t known a priori,
- you want to issue a request to one of several objects without specifying the receiver explicitly, and
- the set of objects that can handle a request should be specified dynamically.

![Handler Diagram](image)

**FIGURE 5.18.** Handlers: [Gamma et. al. 95]

**SP-II Handlers**

Currently, what SP-II has to be able to handle is the generation of minimal FUs to start of a SeedLayout problem as input. The product model of AP in this scenario has to define objects like PointThreeD, Point, Context, BuildingContext, SpatialFU, etc. so that the communication modules can match the objects generated by SP-II to those existing in the database and provide faultin and faultout functionalities. All these are already treated as objects in the Seed Layout domain. In SP-II to generate an FU means to be able to generate a model which has all the information necessary to recognize an SU as an FU.

Seed Layout already has problem specification functionalities. To match the prototype model discussed in section 4.2.1 to the communication schema, the exercise would be to generate a set of objects based on this internal representation. For example, `MakeContext` would generate a Java object reference to the Java objects generated after the compilation of the SPROUT schema through the `SpecCategory` Context model. A `MakeBuildingFU` would have to handle classification group objects based on specific requirements model.
The suggested generic design of SeedPro factories and handlers; however, does not provide any suggestion for how to manage schema extensions of the modeling language environment. A rule based parser could provide a solution if the SP-II’s product modeling abstraction and the capabilities in SPROUT is studied. For that matter the construct could work for any modeling environment. The capabilities provided in a modeling language bring more flexibility compared to programming languages. Rule based algorithms could efficiently handle such generic versus specific relationships. However, such a study involves the exhaustive modeling of the usability and decision support scenarios that come with it (a preliminary study will be presented in Chapter 6). In addition, the intra agent solution spaces are also of concern. For example, an SU containing spatial attributes as a solution is a potential input both for a layout generation, cost estimation and structural analysis modules. The SU of spatial attributes must be handled as a SpatialFU objects for Seed Layout, as a cost specification object for cost estimation, and as a structural module object for structural analysis.

**Primitive Handling**

In SPROUT schema relationship types are containers such as set, link, bag or vector which do not have their corresponding type in the specification environment, as a primitive which can be used in the modeling stage. Thus a generic mechanism to handle relationship types has to be designed. A layout problem in its specification in the SPROUT schema has attributes “relationship context hasPart (Context) inverse problem” and “relationship topUnit hasA (SpatialFU) which when compiled to generate a database schema are sprout link types.

SpatialFU and Context are defined in our prototype model as specification categories, so in terms of an SP-II object the relationship inference has to be made. This task involves high semantic integrity issues both in the context of SEED as a system, and more important and problematically in SP-II as a stand alone tool. SP-II has to be able to manage multiple SU declarations without the need of useless extension of the shared schema decisions.
In writing the handler methods for communication the inheritance and abstraction posed in inheriting methods from SproutObjectHandlers assume that the desired SP-II objects are defined in the module [Figure 5.20.].

5.4.3 Mapping Manager

Mapping manager is the module to access the communication modules through the protocol manager to send the objects. One generic mapping manager is in charge of the database connection.
The mapping manager is responsible for taking care of the communication windows, registering sent and retrieved objects and pass Java to C++ mappings around between SPROUT objects and C++ based agent objects. Currently the objects that SeedLayout module is interested in saving in the database are Functional Units, Layout Problems, SL_Context, Design Units, thus SLMappingManager is written against these.

In the context of SP-II, the objects that SP-II is interested in sending are SUs, which can have other SUs as constituents. Here again the specific schema decisions are in question. In Mappingout a functional unit first the protocol has to identify what kind of an Fu it is, like Roof, FlatRoof, SlopedRoof, ArchZone, Mass. A sent out SU object in SeedProMappingManager has to be able to handle the same queries so that the object specifications get saved in their correct database representations. The object model in Figure 5.21. lists methods related to an SU. To use database support for any object like for saving in products, product models, spec categories, classifier groups, etc. desired such methods must be provided.

### 5.4.4 Language Bindings

![FIGURE 5.22. SP-II Generic Language Bindings Model](image)

```cpp
#include ET++.h
#include SeedProFactory.h
#include GenMechanism.h
#include Product.h
#include ProductModel.h
#include SpecElement.h
#include SpecUnit.h

SPLangBinder

- SPLangBinder()  
- ~SPLangBinder()  
- InitializeSpLangBinder():bool  
- MapOutSu(obj:SpecUnit *,jobj:JavaObjRef *):bool  
- MapOutSpecCategory(obj:SpecCategory *,jobj:JavaObjRef *):bool  
- MapOutSpecElement(obj:SpecElement *,jobj:JavaObjRef *):bool  
- SupMapOutIntegerPrimive(obj:IntegerPrimitive *,jobj:JavaObjRef *):bool  
- SupMapOutFloatPrimitive(obj:FloatingPointPrimitive *,jobj:JavaObjRef *):bool  
- SupMapOutStringPrimitive(obj:StringPrimitive *,jobj:JavaObjRef *):bool  
- SupMapOutBoolPrimitive(obj:BooleanPrimitive *,jobj:JavaObjRef *):bool  
- SupMapOutEnumaratedPrimitive(obj:EnumaratedPrimitive *,jobj:JavaObjRef *):bool  
- MapInSpecCategory(jobj:JavaObjRef *,obj:SpecCategory):bool  
- MapInSpecElement(jobj:JavaObjRef *,obj:SpecElement *):bool  
- SubMapInIntegerPrimitive(jobj:JavaObjRef *,obj:IntegerPrimitive *):bool  
- SubMapInFloatPrimitive(jobj:JavaObjRef *,obj:FloatingPointPrimitive *):bool  
- SubMapInStringPrimitive(jobj:JavaObjRef *,obj:StringPrimitive *):bool  
- SubMapInBoolPrimtive(jobj:JavaObjRef *,obj:BooleanPrimitive *):bool  
- SubMapInEnumaratedPrimitive(jobj:JavaObjRef *,obj:EnumaratedPrimitive *):bool  
- MapOutProductModel(obj:ProductModel *,jobj:JavaObjRef *):bool  
- MapInProductModel(jobj:JavaObjRef *,obj:ProductModel *):bool  
- MapOutGenMechanism(obj:ProductModel *,obj2:ProductModel *,jobj:JavaObjRef *,jobj2:JavaObjRef *):bool  
- MapInClassification(jobj:JavaObjRef *,obj:ClassifierGroup *):bool  
- MapOutRelation(obj:RelationType *,jobj:JavaObjRef *):bool  
- MapInRelation(jobj:JavaObjRef *,obj:RelationType *):bool  
- MapOutProductModel(obj:Product *,jobj:JavaObjRef *):bool  
- MapInProductModel(jobj:JavaObjRef *,obj:Product *):bool
```

FIGURE 5.22. SP-II Generic Language Bindings Model
Language binding methods, using the factory and handler modules described in sections 5.4.1. and 5.4.2 respectively, match the Java objects with those of the internal representations of the agents. If a specific schema is desired to be defined in between agents, these bindings would have to refer to these specific objects. For example, for the Seed Layout objects, referring back to the prototype model, mapping out an FU to the database from SP-II requires mapping out `SpecCategory`, `Classification` and `RelationType` objects repeatedly with specific naming conventions. This approach is possible, but not plausible since it requires hard coding and assumption that the users are aware of the naming conventions required. Moreover, such an approach assumes that no changes to modules are made and no human errors occur in modeling.

Extending the SPROUT schema with SP-II specific objects and writing an internal generation mechanism in between different views of the schema, like Seed Layout view and SP-II view, is more in the spirit of the dynamic modeling approach. This also allows the extension of generative programming notions into their corresponding data models and aids the emergence of collaborative decision support scenarios.

### 5.5 Conclusion

This chapter aimed to specifically describe parties included in the information exchange scenarios in SEED and present implementation design with their drawbacks in integrating SP-II to the system. Since SP-II as a requirements modeling tool is interested in the generation of objects beyond FUs, a generic solution has to be sought for. If requirements model and shared SPROUT schema specific decisions become central the involved design and code has to be repeated for even storing an SU, which is one of the central objects of SP-II. Therefore, the work is left at design stage to extend the implementation with generative, dynamic and generic capabilities for collaborative decision support in requirements modeling. Next section will present some of the preliminary studies done for this purpose.
6. **Preliminary Proposed Framework**

Based on the critique and design presented in Chapter 5, this section present the results of preliminary models to incorporate use cases of a DS, and requirements of information sharing and data support in a dynamic programming framework. The role of requirements modeling is seen more general in this section instead of limiting the communication schema to the SEED modules. The use-case scenarios all assume the support of a database engine.

### 6.1 Introduction

Decision support is a very broad term. Architectural decision support is even broader because of the interdisciplinary and collaborative nature of the problems. The functionalities that could be associated with computational decision making could be numerous, e.g. collaborative scenarios, case-based problem solving and generative programming. However, in terms of information coherence and data support, modeling a generic data engine, which can be extended as new agents are required for computational DS, is possible. In this section such generic capabilities will be discussed.

### 6.2 Context

Our domain of study is broad and ill-defined. Requirements specification is seldom achieved in a complete manner. Usually there are specifications which are intangible and uncountable, yet they still define an end product in terms of performance, form or needs. AP does take this into account, however to computationalize it requires a scale against which such attributes must be evaluated. Architectural decision making is a process of making inferences from both abstract, concrete and tacit data.

The AP defined for the context of SEED and the requirements captured for that matter all include concrete constraint based data. Generation of different views from a set of programmable requirements and those from abstract descriptions both have to be combined in DS. As discussed earlier, knowledge-based approaches provide an alternative, however when the target domain becomes generic as requirements modeling, problem solving approaches must also be based on generic capabilities.
AP has previously been defined as the derivation of design concepts from a program concept. This process of derivation has been the task of the design team traditionally. A computational tool is the replaced platform to assist such collaborative derivative tasks.

The DS module is one that carries multi-functionalities like CBR, consistency and constraint management and generation and reasoning mechanisms. Such problem solving scenarios for requirements modeling need support from a computable scaling of specifications.

### 6.3 Domain Study

Figure 6.23. represents a diagramatic view of the domain of interest. DS module is a central agent where other components meet at. A requirements modeling problem, an AP, is an input to DS. There is a large set of users who could interact with DS to solve a requirements modeling problem. Since a generic and dynamic data modeling approach is taken the outputs of DS, i.e. solutions, are parsed in shared schema and mapping modules and are maintained in a database. An intermediary link as such is assumed to allow multiple view generation of the dynamically defined data set. The specification space is provided as a supportive mechanism to manage ill-defined and specified data capture which frequently occurs in AEC.

#### 6.3.1 User Space: actors

The actors of the decision support module for requirements modeling must be studied in two sub groups.

**Passive actors.**

*Agents:* Other agents that are assumed to be in collaboration with SP-II as a module must be able to retrieve cases relevant to their internal representation. These are designated as passive actors because their core purpose in interacting with the...
data is to retrieve it for their own internal use. The capabilities offered in the context of requirements modeling do not apply in the internal context of these modules. The data management should facilitate such an interaction without any overheads. Therefore, although they do not display active use cases the sequences required by other agents are important features of the overall problem.

**Active actors.**

Both modelers and programmers are professionals who belong to sub disciplines in AEC industry.

*Modeler:* Modeler is seen as the expert who provides the alternative models and generation mechanisms that are to be used for SU production. So far the role of the modeler in SP-II has been secondary. In the context of data and DS functionalities, the design knowledge captured also depends on the modeling process. Modeler, beyond the use cases provided for him in SP-II, could access various product models and generation mechanisms to provide new requirements models. Collaborative modeling has not been addressed either. Mechanisms to support collaborative program generation and multiple modeler scenarios, should also be incorporated.

*Programmer:* A programmer calls a predefined requirements model and fills in the required data. When data support is integrated to the module, the programmer will be able to use prior defined cases, browse and adapt precedent knowledge, define new programs and index them as cases. A programmer is more of an instantiator and data provider.

*Other domain users:* Since a generic and dynamic requirements modeling is pursued it is possible to develop scenarios for other domains which benefit from requirements specification, e.g. industrial engineers, interior designers.

**6.3.2 Specification Space**

Requirements modeling has been previously defined as an ill-defined process which requires management of scattered and multi domain knowledge. In structuring such fuzzy domains an abstraction scale is believed to be useful. A model which is adapted is the concreteness and completeness scale of design protocol evaluations [Eisentraut et. al. 97]. A two dimensional matrix of scaling requirement specifications form incomplete to complete and from abstract to concrete provides patterns along which different states in design can be matched against. As drafting, requirements modeling also has intermediate states which become inputs of latter stages of design. Such a pattern would especially be of use in guiding the designers through alternative paths in DS.

**6.3.3 Usability Goals of the Decision Support Module for Requirements Modeling**

Design is not a linear process, it cannot be assumed that design starts with requirements specification. However, computationalizing design requires some assumptions to be made. Although design process does not necessarily start with AP, the iterative and cyclic nature of design problems allow centralizing problem definition tasks as AP. Goals of a requirements modeling tool, some of which have been addressed earlier, in regards to information management for dynamic frameworks are:
Data support should provide a shared solution repository across different agents, which are related to each other through knowledge transformation.

The tool should serve multi domain users who participate in the requirements modeling process. The role of the modeler is an interdisciplinary one. (architectural programmers who are expert in spatial programming, might not understand mechanical requirements specification)

Information exchange requires communication between users both through the artifact they act upon and through awareness of each other's actions. Achieving this aids in enhancing both synchronous and asynchronous collaboration.

The functionalities provided in relation to the database should not reflect the drawbacks of generic database applications where computer support is minimal and the information structure is not presented in an easily perceptible manner. ¹

Data should be visually managed that is understandable to different domain users: textual representation, graphical representation in 2D or 3D, diagrammatic representation hierarchical representation must all be considered.

The model representation should be transferable between different views so each user in the collaborative work process through the database should understand the same thing meant by another user.

Aid in browsing and queries through robust communication patterns must be provided

Correct translation of operational CBR metaphors such as retrieve, match, adapt, index, delete, anchor to minimize layers of complexity from users' perspective is a central issue. Assistance to users in the application of CBR metaphors and operations should be provided. Users should not need to know the computational process that the tool follows to arrive at the solution.

Integration of several modules like the communication and database servers with a decision support module must be managed.

Only depending on SUs as the smallest unit of information package causes cognitive overhead since each SU has a different configuration based on its host product model. Therefore, specification and classification categories, relationship types and generative programming capabilities based on formula specifications must become a part of the knowledge to be captured and modeled. Such an approach allows development of new models and APs based on prior ones using data retrieval and adaptation capabilities. In addition, this mechanism could support the strategy formulations defined by Sen.

Retrieving different product models, generation mechanisms and SUs should be treated as different case-bases, which depend on each other but could also be treated independently.

¹. In most of the prototypes produced a library functionality was provided. The libraries of SUs provided in SP-I or the libraries foreseen in the technology specifications [Sen 98] were depending on flat saved files. Incorporating these in the database has a lot of advantages, while employing the same library metaphor in GUI.
6.3.4 Core Features

**Multi-tier repository and feedback capabilities:** The users of the dynamic requirements modeling tool should be able to store product models, generation mechanisms between product models and SUs both independently and in relation to each other. The navigation through these sub categories should also be managed accordingly. Aid and design in browsing should be derived from the design process rather than data management requirements.

**Part - whole relations:** To derive new product models, generation mechanisms and SUs from previously encountered ones it should be possible to extract information in parts. This is not achievable if only SUs become the core elements of concern. Combining multiple specification categories and relationship types from multiple models in producing new ones must be supported. This also applies for the browsing and adaptation of SUs. Such an approach opens plausible paths for multiple system requirements modeling and integration in solution generation.

**Capturing classifications and topologies:** Problems of AP do inherit prototypical features due to the nature of building design. Generic specifications which could be extended to typologies can be captured. The ongoing work by Erhan show such features in clinics, army reserve centers and schools. Similarly, there are quite a number of studies in the literature which address housing typologies. The notion of prototypes as a problem solving approach should be incorporated in the DS mechanisms employed.

**Collaborative Programming:** AP is a team work. Synchronous and asynchronous collaboration must be incorporated in DS, otherwise computational support becomes limited to documentary features, hindering benefits of software engineering principles employed.

**Dynamic and generic data modeling:** User customization is an important feature in enhancing the computational tool support in capturing design processes through algorithms. The models which rely on understanding the processes described by experts and hardcoding these become quickly outdated when a new feature is discovered. Systematically modeled generality may not only provide computable DS, but it could also provide models to understand processes.

**Schema extensibility versus schema overlaps:** SPROUT, hence the database schema is extendible. This is one of the main strong capabilities provided in the SEED information exchange scenarios. Therefore, it is possible to provide an extension to the schema for each new SU configuration needed to be saved. The result of such a scenario is in its essence the production of flat tables, which can even be viewed as spread sheets. The information is valid only in the context of the specific model used. The redundancy of data could very easily become an overhead and the capabilities of database support discussed earlier cannot be incorporated. Therefore, overlapping schema extensions and inheritance properties of models must be explored.
6.3.5 Software Engineering Patterns

In designing DS the software engineering challenges are:

- defining the generic data structures of SP-II in a shared modeling environment as SPROUT,
- developing models to manage user defined semantic contradictions, like naming conventions,
- providing multi-classifications of dynamically defined attributes in terms of specification units, product models and generation mechanisms,
- introducing a mechanism of generating multiple views of SP-II outputs without computational overhead (like dynamic mapping of SUs to FUs),
- providing generic knowledge-base capabilities, and
- integrating such complex implementation layers with seamless interfacing principles.

Section 5.4 addresses some of the items listed above. Generic factory and handler patterns as designed in sections 5.4.1. and 5.4.2. carry dynamically defined SP-II objects into the database. However, once requirements specifications are represented in the database as generic objects the next problem becomes defining these in terms of multiple agent objects. The initiative must again be left to the user. A pattern to browse generated SPROUT objects and provide attribute mapping might be the approach. The builder pattern addresses a similar construct and can be employed in generating objects of interest to other agents from requirements models objects.

![Builder pattern diagram](image)

**FIGURE 6.24.** Builder pattern: [Gamma et. al. 95]

The **Builder** pattern is applicable when:

- the creation of a complex objects should be independent of the parts that make up the object and how they are assembled,
- the construction pattern must allow different representations for the object that is constructed.

---

2. The model evolution concept of EDM-2 discussed in Section 4.3.1. describes a similar problem.
The necessity of representing SU hierarchies of generic SP-II objects in terms of Seed Layout (or other agent) objects might benefit from such a construct. The parts would be the SpecCategory, ClassificationGroup and RelationType objects, while the products would be different SPROUT objects. Although this approach would also involve intensive implementation generality and automation would be the gain. The builder pattern would be a third pattern employed to bring database objects that are generated by different applications together.

### 6.4 Use Case Explorations

The use cases required for DS with a dynamic framework are helpful in specifying implementation driven problems of data representation through shared schema rules. Below are some of the preliminary ones which are of core importance.

#### 6.4.1 Libraries - a CBR abstraction

Use of libraries is a metaphor to replace database terminologies. Linking to a library and retrieving an existing data from a library is a common use case employed in various computer tools (like AutoCAD and MicroStation).

In the context of SP-II opening a library window necessitates linking to the Communication and DBServers to connect to the commercial database provided, i.e. UniSQL. The use cases associated with communication servers are **specify database server, specify communication server, specify agent name, connect/disconnect, register interests**. These use cases are the default ones required for any agent to connect to the communication

---

3. The documentation of the work is currently under development. For a preliminary report refer to [Flemming et. al. 2000]
FIGURE 6.25. Model for library layers, in OMT notation

Figure 6.24. illustrates the model of a three party information support functionality. The approach of breaking the knowledge-based support into such three sub systems has quite a lot of advantages for DS in the context of requirements modeling exemplified by AP. A.

- It allows collaborative scenarios to emerge.
- Knowledge-based problem solving support has common features in all three, but necessitate different adaptation mechanisms and algorithms. In such a model sub problems can be defined for all three increasing efficiency and correctness of DS.
- In the current implementation multiple product libraries are assumed. This does not allow generation mechanisms across different products. A more general approach aids in generalizing these solution generation alternatives.
- Adaptation scenarios based on precedents become dispersed into different levels of granularity. SUs give quantitative details and instances of product models, product models define an outline of requirements, and generation mechanisms provide new design scenario generations through functional units or design units. Currently all these are inherited in one project or product library. Treating these separately in data management allows multiple system integration and alternative scenario generation.

Both the modeler and the programmer will be interacting with the libraries for different purposes. Basic functionalities associated with a repository support like save, delete, update, copy, retrieve are needless to be discussed. They are assumed to be present and will not be further elaborated. The mechanisms and patterns facilitating synchronized and heterogeneous domain interactions generate the interesting use-case scenarios.

Automated DS through knowledge bases are usually provided with expert systems where the underlying decision mechanisms are extracted from domain studies and are hard coded. In an interdisciplinary environment where there is not one expertise area the high level computational support should benefit from recognition and interaction with users. To be more specific, in SP-II conceptually it is possible to generate two different output models for SeedLayout and SeedConfig from the same requirements package. It is further possible to use these two output models for generating yet another scenario. Handling such incremental scenarios requires a data support engine which is capable of processing multiple levels of queries and classifications. However, it should appear to the user with seamless cognitive metaphors and the process should be modeled generically so that the implementation overheads discussed are handled.

4. The classification mechanisms provided by Aygen make such sub groupings possible. It is possible to retrieve the same information in several sub categories. The essence of the library classifications will eventually have to benefit from Aygen’s classifications since overall the information capture in all depend on the Specification, Classification and Relationship objects currently implemented in SP-II.
In an AP during early stages of design it is as important to know some of the preliminary structural and mechanical decisions as well as composition of spaces. Such specific data usually is provided only at construction stages where drawings are the communication medium. Therefore, a more fluent specification environment where simultaneous data exchange is beneficiary. An example scenario would be while the structural engineer develops the necessary AP and aggregates the data to the current requirements model, the interior designer would work on the furnishing. In the life cycle of a firm these new models would become statically established in time, new cases would emerge as new projects instances. The AP computation tool must be able to support this, however so far these issues were not addressed due to the layout generation focus of the project.

FIGURE 6.26. Integration Sequence for Library Selection, using Rational Rose

A sample interaction sequence of initiating information support from the database through the libraries is given in Figure 6.26.

Instantiating a specific library of generation mechanism, product or SU, is a way of classifying the information and the role of the user. An SU library, hence the cases associated with SUs, are of interest to programmers. They are interested in specific data like the office hierarchies of an administrative unit, or the structural comparisons of a steel high-rise and RC building. On the other hand, selecting a product library would necessitate providing capabilities like viewing and selecting specification categories, classification groups and relation types independent of the product model that host them. Even

5. The object model of SP-II is taken as an example here.
Preliminary Proposed Framework

6.4.2 Generation Mechanisms

Generation mechanisms support is a one that should combine the functionalities of both product modeling and specification units. Moreover it must host functionalities that provide associations between two (or more) product models to customize outputs. Reasoning mechanisms embedded in formula specifications require the associations of both multiple product models and SUs. A dynamic environment should support generality and learning patterns to emerge through output customizations.

6.4.3 Product Modeling

The rationale of having a products knowledge-base would be:

- to select a product model for the instantiating new SUs, this could also be associated with the notions of typology classifications
- to generate new product models from old ones
- to extend existing model specifications

It is proposed that a product modeling knowledge support module should include functionalities as:

*Aggregate*: A use case to build part-whole relations between products

*Associate*: A use case to specify one-to-one, one-to-many, many-to-one, many-to-many relationships

*Classify*: This is a core functionality that would allow high level query specification like classifying the data as product models, specification categories, classification groups or relationships or provide typologies like museum buildings, buildings with atriums, etc.

*Specify Constraints*: This is a core problem in such requirements management systems, constraints are generated in modeling stages and consistency checked during specification.

Such functionalities could become supportive in collaborative decision making scenarios. The scenario modeled here is simple one, yet presents a lot of problematic issues like constraint and version management, awareness of each others actions in team tasks, dynamic semantic integrity, awareness of domain and expertise, and iterative nature of design process. Referring back to Table 3.1 the procedures prescribed in AP do in fact provide methods that could aid in modeling some of these issues. The category of organizing information search is one of the main areas where computational DS in requirements management could utilize. Similarly analyze of information comes under the generative programming capabilities that such tools offer.

Therefore, use cases associated with each information view modeled in Figure 6.25 become specialized.
Figure 6.27. exemplifies a collaborative product modeling action sequences.

FIGURE 6.27. Synchronous collaborative modeling - DB interaction, using Rational Rose

6.4.4 Specification Units

Specification units are of interest to programmers. Requesting SUs as cases require the specification of the host requirements model they are associated with and querying the model to match the new requirements set. The functionalities associated with such a scenario are simple detection algorithms like parsing the specific data that each SU holds and evaluating the SU hierarchies accordingly.

An SU based DS necessitates functionalities such as match and adapt. Ideally it should allow SUs to be retrieved from other host requirements models and adapting those to new cases. All of such capabilities require the well-formedness of the data support in a shared information exchange environment.

Figure 6.28. models the action sequences of a programmer in retrieving and updating an SU.
6.5 Conclusion

The aim of this section was to present some of the preliminary design models pursued to integrate intelligent DS mechanisms with seamless user interaction. It is proposed that any DS tool in the domain of requirements modeling must be customized in reference to task the involved, like AP modeling and SU instantiating are independent tasks. The functionalities provided must be designed accordingly. Knowledge-based support and inference must depend on dynamic and generic mechanisms which change as new standardization and requirement specifications emerge rather than static models.
7. Conclusion

This chapter contains contributions and possible future research directions based on the findings presented before.

7.1 Contributions

The contributions of the study are:

6. Design of integrating SP-II into the information system architecture of SEED
7. Study of the implementation based drawbacks of various modules of SEED (namely SPROUT, SEED database, SP-II and Seed Layout as current parties in a collaboration scenario and the communication servers) in information exchange.
8. Comparison of predetermined schema specific data models and generic models in providing data support for a dynamically programmed requirements modeling tool.
10. Preliminary models for generating multiple views from generic objects in terms of schema specific agent based objects in a bi-directional information exchange scenario.

7.2 Future Research

Data modeling and standardization efforts in AEC industry have given quite a number of unsuccessful research tools during the past few years. Areas of concern like object oriented data modeling, collaborative design, developing generic building representations are among those which are repetatively refered to. The information exchange through the multi agent system architecture of SEED also brings most of the current professional concerns to attention as potential research areas.

Dynamic Data Modeling.

The core of this study was devoted to addressing difficulties of achieving dynamic data representation for requirements modeling. A model inheriting from software engineering principles could provide a plausible framework. Issues of well-formedness and
Conclusion

semantic integrity of building representation attributes in AP form the challenges of such a research. However, the question is worth the time since the field of architecture lacks computational tool support in a generic format and research efforts cannot catch up with technological advances.

**Constraint Management and Consistency Checking.**
Requirements models provide a medium where both tacit and concrete data is captured. Constraint management and consistency checking is a problem which can depend on computational support. Such efforts would contribute to achieving semantic inferences through computer support which usually human beings are much better at. Moreover, algorithms and mechanisms to structure constraint data are also beneficial to standardization efforts in AEC.

**Case-based DS.**
CBR in a dynamically defined attribute domain is a contradictory problem, since CBR is bound to specific problem solution instances. However, a generic platform could benefit from learning mechanisms and genetic algorithms to produce alternative solutions for a broad domain like AEC. The specification categories and classification groups can be treated as the minimal generic information packages and alternatives could be generated.

**User Interface modeling for dynamic data management.**
HCI has become a research domain which provides approaches to depict ill-defined problems through user models. Dynamic requirements modeling can benefit from HCI principles in structuring the generic information in multiple platforms. A potential question is how to design an interface to guide the user in browse of the unspecified SU and product model data. An other one is how to model problem solving of a DS module.

**Collaborative Requirements Modeling.**
AEC is a multi-disciplinary domain. The nature of production necessitates team work and most of the time use of multiple domain specific applications. Therefore, distributed design, asynchronous and synchronous use of data and information are not only data management problems, but they are also design process questions.

**Generative Programming.**
AP as a requirements model presents a formalization of scattered information towards the generation of specified functional and design units. Each view of an AP is another solution space. Reasoning mechanisms to arrive at these solutions lack a structured and well-formed established process.
References


References


References


[Kolodner 93] Kolodner J. L. (1991)”Improving Human decision Making through Case-Based decision Aiding” *AI Magazine* 12(2) pp.52-68


References


[Pree 95] Pree W., (1995), Design Patterns for Object Oriented Software Development, Addison-Wesley NY


<table>
<thead>
<tr>
<th>Reference</th>
<th>Citation</th>
</tr>
</thead>
</table>