DESIGNING DOMAIN-SPECIFIC HUMS ARCHITECTURES: AN AUTOMATED APPROACH[1]

Ravi Mukkamala[2] *Neha Agarwal* *Pramod Kumar* *Parthiban Sundaram*

Department of Computer Science

Old Dominion University

Norfolk, VA 23529

Abstract

The HUMS automation system automates the design of HUMS architectures. The automated design process involves selection of solutions from a large space of designs as well as pure synthesis of designs. Hence the whole objective is to efficiently search for or synthesize designs or parts of designs in the database and to integrate them to form the entire system design. The automation system adopts two approaches in order to produce the designs: (a) Bottom-up approach and (b) Top down approach. Both the approaches are endowed with a suite of quantitative and qualitative techniques that enable a) the selection of matching component instances, b) the determination of design parameters, c) the evaluation of candidate designs at component-level and at system-level, d) the performance of cost-benefit analyses, e) the performance of trade-off analyses, etc. In short, the automation system attempts to capitalize on the knowledge developed from years of experience in engineering, system design and operation of the HUMS systems in order to economically produce the most optimal and domain-specific designs.

1. Introduction

Engineering disciplines progress through several distinct stages in their evolution: Ad-hoc, Formal and rigorous, and finally, Automation [1]. This work focuses on advancing HUMS to the automation stage since HUMS, as a discipline, is mature and well established now with various systems already deployed and in existence for a long period. The automation system so developed would attempt to capitalize on the knowledge developed from years of experience in engineering, system design and operation of the HUMS systems in order to economically produce the most optimal and domain-specific designs.

The design of software systems has largely been performed manually so far. The challenge in the automation of system designs is (a) the development of design methodologies that support automation, (b) the accumulation of knowledge – components that make up design, trade-offs involved, decision making criteria, requirements, etc. – and also (c) the provision of means to evaluate the quality of the designs. The automation, if successful, would be highly beneficial in the following ways: (a) minimization of costs involved, (b) improvement in productivity, and (c) production of optimal and innovative designs, etc.

Our automated design process involves two main techniques: (a) Selection of solutions from a large space of designs, and (b) Synthesis of designs. However, the automation process is not an absolute Artificial Intelligence (AI) approach though it uses a knowledge-based system that epitomizes a specific HUMS domain. The process uses a database of solutions as an aid to solve the problems rather than creating a new design in the literal sense. Since searching is adopted as the main technique, the challenges involved are (a) to minimize the effort in searching the database where a very large number of possibilities exist, (b) to develop representations that could conveniently

[1] This work is supported in part by a research grant from NASA Langley Research Center, Hampton, Virginia.

[2] Contact author: Email: mukka@cs.odu.edu

allow us to depict design knowledge evolved over many years and (c) to capture the required information that aid the automation process.

Two groups of participants are involved in this process: Sources and Developers. Sources group includes end-users, domain experts, requirements analyst, etc. and the Developers group involves requirements analyst, system developers, etc. Due to the nature of the data required by the automation system and the partial reliance on humans for decision-making, the developers group would be more successful in interacting with the automation system.

There are three basic design approaches that the automation system could adopt such as: (a) Bottom-up approach, (b) Top-down approach and (c) Hybrid approach. Bottom-up approach depends on a hierarchical partitioning of the system into multiple levels and builds the entire system design from scratch starting from the bottom level up. On the contrary, the top down approach first starts with an entire system design and then drills down to lower levels by searching for the individual component instances that can fit the design. In this paper, we shall focus on bottom-up approach since it is easy to follow, and a large majority of its techniques are common among the approaches.

The bottom-up approach is a five-step process involving the following:

- Capturing Requirements Data
- Capturing HUMS component information
- Designing the architecture
- Evaluating the candidate designs
- Producing the System Output: Architecture Specification

The rest of the paper explains in detail how each of these steps is implemented. Section 2 discusses the related previous work by others. The automation system is made up of a number of building blocks, which are explained in Section 3.Sections 4 and 5 explain two of the three input resources used by the automation system namely the Requirements Information and the Components information respectively. The selection of HUMS components for the architecture and the determination of their configurations are explained in section 6. The automated design process, the roles played by different building blocks of automation system and the evaluation of candidate designs are described in section 7. Finally, section 8 concludes the paper with a summary and future work.

2. Related Work

Software architecture has more influence on the system's quality attributes than the code-level practices [7]. The advantages of architecture as a high-level abstraction were realized and the different techniques to design architectures and to evaluate architectures emerged [4, 14,15]. This work also resides at the architecture-level, thus working at an abstraction level that allows analysis of designs even without accurate or low-level details.

The Architecture Tradeoff Analysis method (ATAM) [6,7] is a technique that helps the designer to perform trade-off analyses in a principled manner thereby enabling him to make well informed as well as optimal tradeoffs. ATAM also incorporates a process model that an organization must follow while developing architecture. Our work is closely related to ATAM in that it also performs tradeoff analyses during the synthesis of architecture but it focuses largely on automation of tradeoff analyses with very little user intervention.

Attribute-based Architectural Styles (ABAS) [8] are architecture styles that are endowed with a reasoning framework useful in analysis of the style with reference to a specific attribute of interest. A given ABAS focuses exclusively on a single quality attribute and thus one needs to consider multiple ABASs to satisfy several quality attributes. This work takes advantage of all the work performed by quality attribute communities and provides us a formal framework to analyze and make design decisions.

SAAM is an evaluation method that uses scenarios in analyzing architecture for different quality attributes [9]. The architecture quality is analyzed by measuring the extent of code modifications required to implement a scenario. SAAM is well suited to be used during implementation stage, while ours is appropriate to be used during design stage to uncover the

problems with the architecture and to improvise it before implementation.

Automation of design processes has been researched and developed in computer systems as well as wide variety of other disciplines [3, 12]. In [3], the authors explain and demonstrate how automation can play a vital role in success of design engineering but their focus is at a lower abstraction level than that considered in our work.

Much work is underway on the standardization of architectures for specific HUMS functions or domains [10,11]. Non-proprietary standards can promote an environment where end users have multiple choices that are economical, as well as high in quality. Our work coexists with and upgrades this work by incorporating lot of elements from the standard architecture like functional layers involved, interfaces that need to be supported, etc. and by adding additional value by enabling analysis and development of system architectures based on the standard architectures published.

3. Automation System Building Blocks

Figure (1) shows the basic building blocks of the HUMS design automation system. The blocks bounded by the dashed lines represent processes while the other blocks indicate information repositories.

The four information repositories are: (a) Requirements, (b) Components library, (c) Design Space, and (d) Design Specification. Requirements block contains all requirements information supplied by the user. Components library is a repository of component instances supplied by several vendors. Design space is a database of designs, which may be templates that can be instantiated to satisfy specific needs, or specific designs themselves. Design Specification block stands for the final design representation that is generated by the automation system at the end of its execution. The automation system must satisfy two requirements while generating the design specification: (a) Design justifications must be provided at every stage, and (b) Architecture must be specified using graphical and textual forms [4, 13].

The two processes are: Explorer and Evaluator. Explorer explores different combinations, or searches for solutions at every stage of the design process and comes out with a set of candidate design solutions that solve the problem at hand at that level. Evaluator checks or evaluates the candidate architectures at every stage, eliminates the bad candidates and retains the best ones. Thus, explorer and evaluator work closely – back-to-back. Explorer completes one step, interacts with evaluator to narrow down the possible candidates and then progresses on to the next stage. This process continues till the entire system design is completed.

Some of the exploration or reasoning techniques that could be part of Explorer are given below:

- Compare the requirements data against the parameter values (Direct Comparisons, Indirect Comparisons, etc.)
- Perform Cost-benefit analyses
- Perform Trade-off analyses
- Use a criteria list or design guidelines in the selection of components
- Search the designs database for a fitting solution
- Use engineering/formal techniques or Quantitative design techniques
- Use heuristics to solve complex design problems
- Others

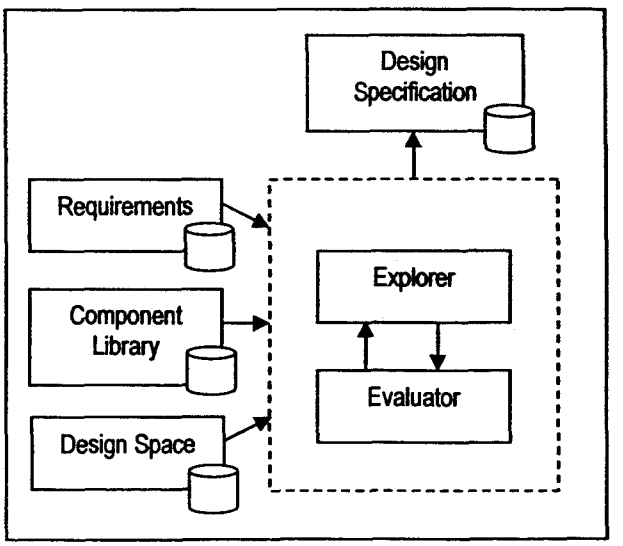


Figure 1: Automation System Building Blocks

Similarly, the techniques that could be packed within the Evaluator process are listed as follows:

- Use evaluation metrics to ascertain quality (a) at system level, (b) at component level
- Build a pareto-optimal set to eliminate bad candidate architectures
- Use heuristics in evaluation

4. Capturing Requirements Information

Irrespective of the design approach adopted, the requirements must be specified in a way that enables the automation system to choose the most optimal design. The general tasks that are involved in any design process drive how the requirements information must be represented. Some of the general tasks are direct comparisons, cost benefit analyses, tradeoff analyses, etc.

The way we represent requirements information must allow the ready employment of at least the most general strategies/tactics listed above. Let us now look at a few objectives that a representation should satisfy:

- All details (Quantitative and Qualitative) pertinent for the design process must be captured
- Must support a format that allows qualitative analysis also
- Data and units specified must be generic enough to allow selection from among component instances of different types
- Whenever data is inaccurate, the associated degree of uncertainity must be specified
- Future growth/usage profile must be captured wherever appropriate

We have developed requirements templates based on XML that defines all the pertinent information that must be captured during the requirements gathering phase. These templates enable developers to gather enough information and in correct form so as to perform effective designing. One difference from the regular requirements analysis process is that here the details gathered are a mixture of 'what' and 'how' information as against the usual practice of recording 'what' information alone. The use of XML in the definition of templates provides better organization and extensibility options. Refer to Appendix A for a sample requirements representation.

5. System Decomposition/ Components

System decomposition is essential for both top-down approach and bottom-up approach. A HUMS system is decomposed into components. E.g.: Sensors, Nodes. Multiple vendors exist who can supply these components and a vendor-supplied component is called component instance. Thus, there exist multiple component instances for a given component.

Table 1: System Decomposition Adopted

Component	Description
Sensor	Senses the measurement parameters
Transducer	Converts any type of input signal to electrical signal appropriate for the nodes
Sensor-Node bus	Connects the sensors to the computing node
Node	Computers/Nodes where all processes reside
Network	LAN, WAN or Internet connecting all the nodes
Processes	Processes that implement HUMS functionality

The design approach comes out with a system design that describes what components make up the system and how they are interconnected. It also specifies the subset of suitable component instances that can be used for the components specified as part of the design.

The HUMS system could be decomposed in multiple ways. So, the most important requirement is that the specific decomposition chosen must (a) accelerate the selection of best components, and (b) ensure the attainment of quality attributes. We have chosen the most fundamental decomposition possible that gives the maximum flexibility in using the components in composing system designs. Table 1 depicts the system decomposition into various components.

6. Selection of Components & their Configurations

This section briefly describes how the lower level components like sensors, transducers, nodes, etc. are selected and how their configurations are determined. Once the nodes have been selected, the design technique explained in section 7 can be used to map software architecture onto these nodes.

The user specifies the following sensor information: (a) Location of sensors, (b) Type of sensors, (c) Other constraint information related to sensors, etc. This information directly allows us to select sensors. The rest of the components like transducers and nodes are determined based on five factors namely: (a) Bandwidth limitations, (b) Buffer limitations, (c) Sensor, Transducer topology, (d) Processing Speed, and (e) Storage requirements. However only those component instances that pass the evaluation by weight heuristic are considered. See Appendix D for sample weight heuristic.

Bandwidth Limitations:

Sensors produce data at a specific rate and multiple sensors are connected to the same consumer (transducer or node). However, the number of sensors that can be connected to the same consumer depends on the external I/O Bus capacity, besides other factors. Our work uses a greedy algorithm to assign multiple sensors to transducers and then multiple transducers to nodes by maximizing the bandwidth utilizations. For example, if sensors with data rates of (in units/sec) 2, 3, 7, 10 and if transducers with maximum capacity of 15 units/sec are present, two transducers would be chosen to connect to the 4 sensors—one receiving 15 units/sec and the other 7 units/sec. This approach thus minimizes consumers required.

Buffer Limitations:

Note that not all sensors that were mapped to a transducer based on bandwidth considerations can be supported because the limited buffer space available at the transducer further restricts the number of sensors connected. The same argument applies to the connection of transducers to nodes. We alter the sensor-transducer mapping and the transducer-node mapping based on buffer space as the second step. This problem is mainly due to the periodicity of the producers: Some producers are periodic and others are aperiodic as shown in Figure (2). For aperiodic case, the consumer must satisfy at least the condition that the buffer space available must be greater than/equal to twice the peak data rate.

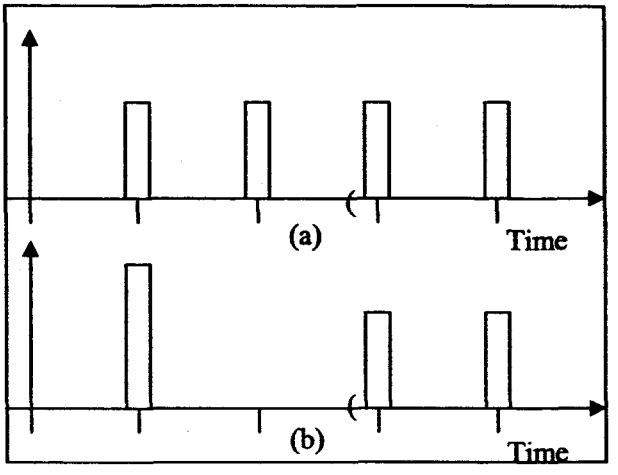


Figure 2: (a) Periodic Sensor (b) Aperiodic Sensor

Sensor, Transducer Topology:

The distance over which a sensor can communicate effectively affects the position at which a consumer is located. Thus, the signal attenuation in the medium is an important consideration. Our work determines the positions of the consumers based on several factors like the power levels (P_{in}, P_{out}) of the transmitted signals and the attenuation coefficients (τ) of the buses involved. For e.g., using fiber optic buses, the distance of separation, $d = \frac{1}{\tau}(10\log_{10}\frac{P_{out}}{P_{in}})$

Node Configuration:

The configuration of the nodes like processing speed, memory capacity, disk space must be determined using some reliable techniques. We make use of custom benchmarks designed for specific domains or existing benchmarks in our decision-making. Such benchmarks could give us quantitative information related to multiple factors involved thereby allowing us to determine the node configuration. Table 2 displays the sample benchmark data that would help us to determine

disk space, bus bandwidth, processor speed, etc. assuming that all operations could be expressed as transactions in HUMS systems. Refer to [16] for a more detailed account. Note: TPS stands for Transactions per second.

Table 2: Relationship between TPS and file size

TPS	File Size
10	0.1 GB
100	10 GB
1000	100 GB

7. The Automated Design Process

A designer relies on the design knowledge that has been accumulated over several years to guide in his design task. This knowledge captured in the form of principles and guidelines could minimize errors and enable achievement of quality attributes. The design space or design library (see Figure 1) of the automation system stores this design knowledge in form of design templates called base models.

A base model is like an architectural style [8, 15] containing information about the components that make up the architecture and their topology. These models also possess quality attribute models (formula sections), which enable us to determine the effect of the base model on the specific quality attributes. These quality attribute models are similar to those discussed in [8] but their purpose in this design technique is to enable analyses as well as automation.

The design space consists of several such base models and each model guarantees one or a few quality attributes alone. If base model '*A*' serves to improve quality attribute '*B*', then model '*A*' disregards all quality attributes except '*B*'. Thus, the base models tend to skew the final design with respect to the quality attributes that they deal with. Figure 3 shows the specification of one of the base models called Flat Model that improves performance.

Quality attributes are often in conflict with each other. Achieving one quality attribute often comes at the cost of another. For example, design decisions favoring good performance affect scalability, those favoring scalability affect availability, and so forth. Resolving between these competing set of quality attributes requires tradeoff analysis. This automated approach hinges on tradeoff analyses to develop the final design.

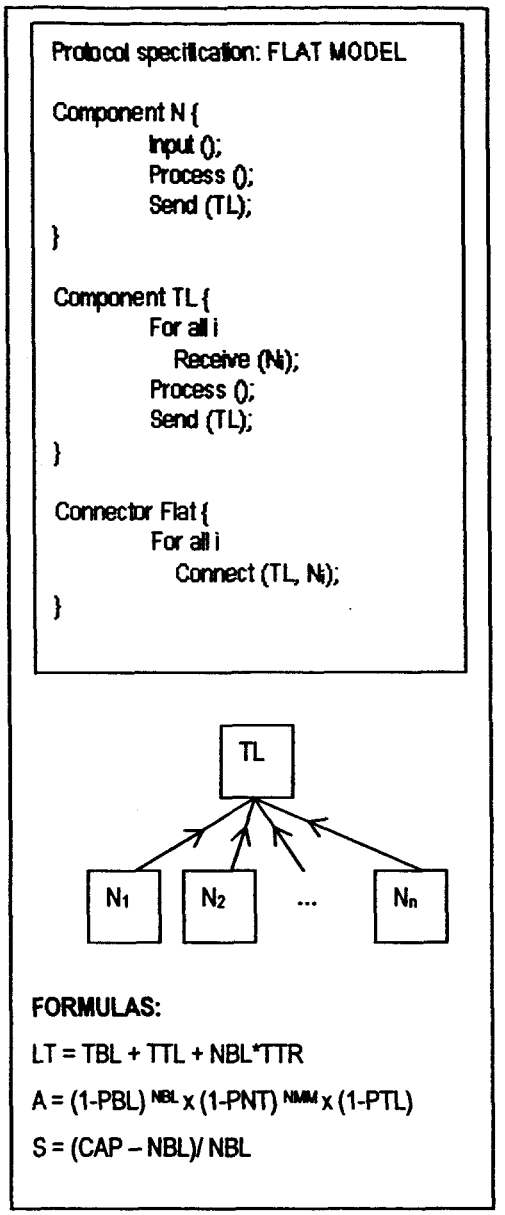


Figure 3: Base Model Specification: Flat Model for performance

Since the base models tend to skew the final design towards one or a few quality attributes, the Explorer must merge multiple models together to develop a suitable design. So, we need two different specifications: (a) Base model specification to specify the architectural style and the quality attribute that is favored, (b) Merge specification that specifies how two different base models can be merged with each other. These specifications enable

the automation system to consider one base model after another and merge if needed on the fly to develop the final design.

A base model specification (see Figure 3) contains three sections: (a) Protocol specification that explains the components, their responsibilities and the interconnections, (b) Topology that pictorially represents how the design is organized, and finally, (c) Formulas which are the quality attribute models built for analyzing the effect of the model on different quality attributes. Quality attribute models shown are simplified for demonstration purposes but can be made realistic and in fact, sophistication can be built into them with the help of experts from several quality attribute communities.

In this paper, Formula sections present three quality attribute models: Performance (Latency time), Scalability and Availability. Latency Time is a function of processing time at the nodes involved and the interprocessor communication time. Scalability is expressed as a function of address capacity of top-level node while availability is a function of reliability probabilities of all components that make up the system. Here's a quick run through the symbols used in the formulas section: LT – Latency time, A – Availability, S – Scalability, NBL – No of bottom level nodes, NMM – No of networks, TTL – Processing time of Top level node, TBL – Processing time of bottom level node, TTR – Interprocessor communication Time, PBL – Probability of failure of bottom level node, PTL – Probability of failure of top level node, PNT – Probability of failure of network, CAP – Address Capacity of Top level node. See appendix B for a few more sample base model specifications.

Analysis of the model:

Latency is the time difference between the instant when the bottom-level nodes read sensor data and that when the top-level node outputs the results.

Latency = {Time to process data at Ni} +

{Time to transmit data by each Ni} +

{Time to process data at top-level}

Since the introduction of any intermediate levels between the bottom level and the top-level will add more cost components to the above formula, intuitively this model supports the design guideline for improving performance.

Merging Designs

If there are n base models, then the Explorer has two approaches:

- Try all (n!) combinations possible till a satisfactory design results
- Merge the models in any order that minimizes the number of combinations

Since the merger of two models is not a straightforward step, we provide another set of specifications called Merge Specifications to aid the automation system in its operations. Thus, merge specifications exist for merging any two base models together.

These merge specifications will enable the explorer to combine any combination of base models, thus leading to $m = nC_2 + nC_3 + \ldots + nC_{n-1} + nC_n$ designs. The responsibility of the designer is to feed n base model specifications and nC_2 merge specifications and the system can automatically explore and possibly develop m designs. As can be seen, the value reaped due to automation can be tremendous for higher values of n.

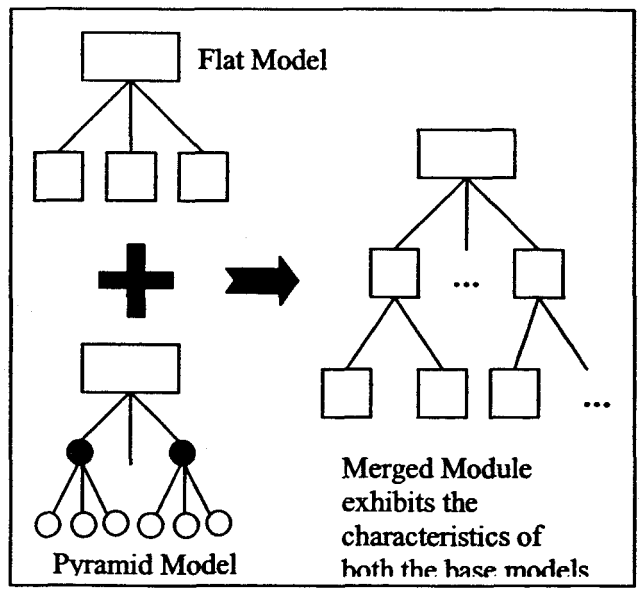


Figure 4: Demonstration of merge process

These merge specifications must consider the impact of different quality attributes that conflict

with one another. For example, increasing the number of levels gives better scalability but yields poorer performance and similarly, increasing the number of replica managers increases availability but decreases performance, and so forth. This emphasizes the need for trade-off analyses. Let's first take a look at merge specifications and then we'll consider the trade-off analyses.

Figure 4 displays figuratively how different base models are merged with one another. As the figure shows, the merger of Flat Model & Pyramid Model results in a design which is influenced by two conflicting forces: Flat Model tends to lower the number of levels while Pyramid Model tends to increase the number of levels. The number of levels that should be in the final design is determined by performing tradeoff analysis. Appendix C shows a few sample merge specifications.

Tradeoff Analysis

Every iteration of merging of base models results in structural changes, which impact the several quality attributes involved in different ways. The tradeoff is achieved in an interactive manner, i.e., the user controls the merging so as to achieve the desired benefits. Of course, the tradeoff analyses can also be completely automated if all the required information is stored.

Demo run:

Tradeoff analysis is explained with the help of a demo run. Table (3) shows the requirements information that contains three quality attributes for this demo.

Table 3: Sample Quality Attributes requirements

Quality Attribute	Requirement
Performance	160 ms
Scalability	1.8
Availability	95%

First, the performance model namely the flat model is used to achieve the required performance. Since this model favors performance, all other quality attributes except performance are disregarded as clearly depicted in Figure (5) & Table (4). The values of quality attributes are determined dynamically using the quality attribute models from the formula section. (See Figure 3) Latency time requirements are satisfactorily met but both Scalability and Availability requirements are not met. Note that the normalized latency time values are displayed in the graph and the corresponding actual values are displayed in table 4.

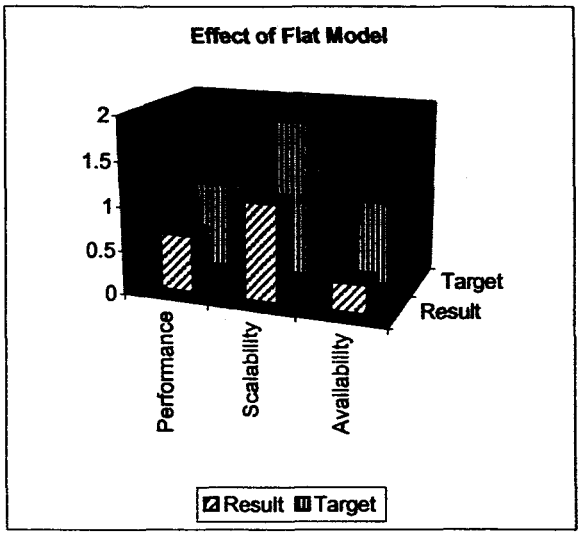


Figure 5: Effect of Flat Model

Table 4: Effect of Flat Model

	Performance (ms)	Scalability	Availability
Target	160	1.8	0.95
Result	105	1.11	0.33

During the second iteration, the automation system searches for a base model that can improve scalability. Now the objective is to achieve both performance (that was obtained during the last iteration) and scalability.

Table 5: Effect of Merging Flat Model with Pyramid Model

	Performance (ms)	Scalability	Availability
NIN=2	130	1.11	0.268
NIN=3	140	2.22	0.2414
Target	160	1.8	0.95

During this demo, the system merges the two base models – Flat model and Pyramid model, and the results are as shown in Figure (6) and Table (5).

Values are computed using quality attribute models in the base model specification (See Appendix B).

As can be seen, as the internal nodes are increased, the scalability increases but affects both performance and availability. However, the performance requirements are met even though it is takes higher value for NIN = 3. Thus, with this merging, both the performance and scalability requirements are met. The next iteration focuses on achieving availability. Note that figure 6 shows normalized latency times while Table 5 has actual values.

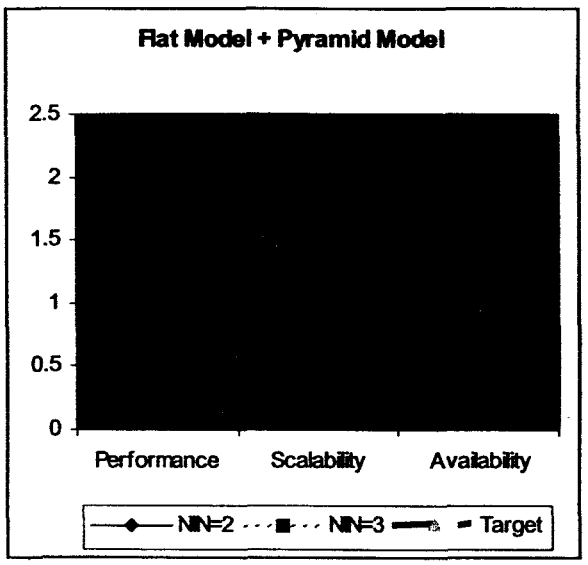


Figure 6: Effect of Merging Flat Model with Pyramid Model

The third iteration starts by a search for a base model that can guarantee availability. During this demo, a base model called Gossip model was chosen and was merged with the existing design (that resulted from merging Flat model and Pyramid model). The results are shown in Figures (7,8) and Table (6).

The user can control the synthesis of the design. After each merger, the data regarding the quality attributes is displayed to the user. The user uses the data to understand the tradeoffs involved and makes the final decision about whether or not another design iteration is required. For example, he may choose the design after the 3rd iteration (See Figures 7, 8) with NIN =3 and NRM = 3, 4 or 5 based on his interests.

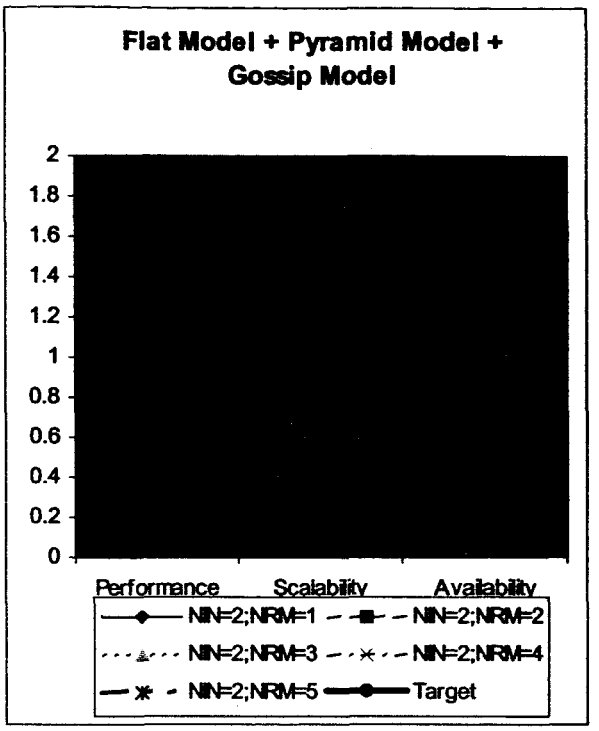


Figure 7: Effect of merger of Flat Model, Pyramid Model and Gossip Model (NIN = 2)

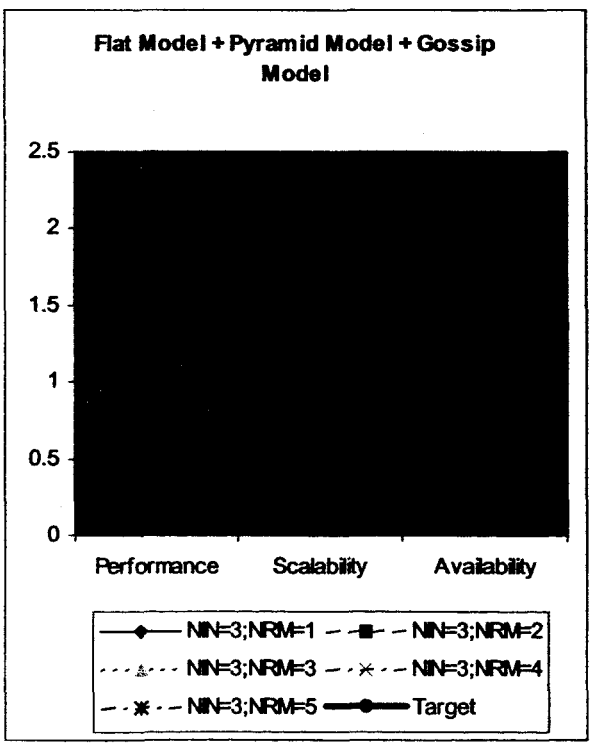


Figure 8: Effect of merger of Flat Model, Pyramid Model and Gossip Model (NIN = 3)

Table 6: Effect of merger of Flat Model, Pyramid Model and Gossip Model

NIN	NRM	LT	Scalability	Availability
2	1	140	1.11	0.2683
2	2	150	1.11	0.8421
2	3	160	1.11	0.9387
2	4	170	1.11	0.9489
2	5	180	1.11	0.9499
NIN	**NRM**	**LT**	**Scalability**	**Availability**
3	1	150	2.22	0.2683
3	2	160	2.22	0.8421
3	3	170	2.22	0.9387
3	4	180	2.22	0.9489
3	5	190	2.22	0.9499

8. Conclusion:

Techniques for design, design automation and evaluation have been developed. The design automation technique can automatically produce (n!) designs for n base models. Base models have been greatly simplified for the proof-of-concept study and sophistication must be added to represent real time behavior. The mean values used for calculations and the benchmarks used have to represent domain information.

Appendix

Appendix A: Sample Requirements

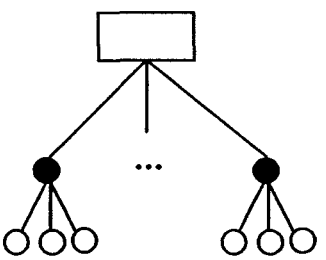
Table 7: Sample Requirements Information for Sensors

#	Parameter	Allowed values
1	Function	Stress, Strain, Temperature, Pressure, etc.
2	Type	Fiber optic, piezoelectric, etc.
3	Data Read operation requirements	
	(a) Medium	Wired Wireless
4	Networking	SINA Wireless No networking
5	Measurement range/value spec	Absolute values or Range

Appendix B: Base Model Specifications

```
Protocol specification: PYRAMID
MODEL

Component Node {//Bottom-level
        Input ();
        Process ();
        Send (Parent);
}
Component Intermediary {
        For all i
          Receive (Childi);
        Process ();
        Send (Parent);
}
Component TL {
        For all i
          Receive (Childi);
        Process ();
        Output ();
}
Connector Conn1 {
        For all Childi
          Connect (Intermediate,
                   Childi );
                OR
          Connect (TL, Childi );
}
```



FORMULAS:

LT = TBL + TTL + TIN + (NBL + NIN) x TTR

$A = (1\text{-}PBL)^{NBL} \times (1\text{-}PNT)^{NMM} \times (1\text{-}PTL) \times (1\text{-}PIN)^{NIN}$

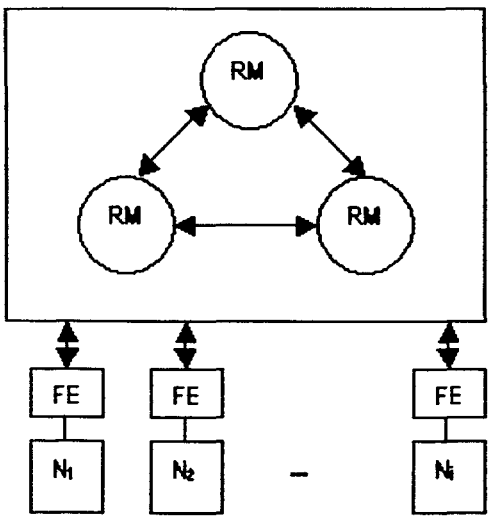
S = (CAP x III – NBL)/ NBL

NOTE: III: Iterator (Current number of levels considered)

Figure 9: Base Model Specification: Pyramid Model for Scalability

```
Protocol specification: GOSSIP MODEL

Component Node {//Bottom-level
        Input ();
        Process ();
        Send (RM);}
Component RM {
    For all i in Assigned[1]
        Receive (Ni);
    For expected number of gossips
        Receive (Gossip);
        Process ();
        Output ();}
Connector Conn1 {
        For all i
          Connect (RM, Ni); }
Connector Conn2 {
        For all i, j
          Connect (RMi, RMj); }
1. Equal set of nodes are assigned to
each RM
```



FORMULAS:

$LT = TBL + TTL + NBL \times TTR + NRM * TGS$

$A = (1\text{-}PBL^{NRM})^{NBL} \times (1\text{-}PNT)^{NMM} \times (1\text{-}PTL^{NRM})$

$S = (CAP - NBL)/NBL$

NOTE: TGS: Gossip Time per unit data

Figure 10: Base Model Specification: Gossip Model for Availability (See [17])

NOTE: All specifications are implemented in XML.

Appendix C: Merge Specifications

```
Merge specification: FLATPYRAMID
MODEL

Change Component {
      Modify (FLAT.N, PYRAMID.Node);
      Create (Intermediary);
      Modify (FLAT.TL, PYRAMID.TL);
}

Change Connectors {
      Modify (Flat, Conn1);
}
```

Figure 11: Merge Specification: Flat Model + Pyramid Model

NOTE: All merge specifications are implemented in XML. Only a sample representation is shown in Figure (11) due to space constraints.

Appendix D: Weight Heuristics

Determination of number of Transducers & their positions:

1. Find $BW_{max} = \max (BW_{Trans1}, BW_{Trans2} \ldots BW_{Transn})$
2. Find $BUF_{max} = \max (BUF_{Trans1}, BUF_{Trans2} \ldots BUF_{Transn})$
3. Find $WT_{min} = \min (WT_{Trans1}, WT_{Trans2} \ldots WT_{Transn})$
4. For every type of sensing required {

For every sensor, sj of the selected sensing type {

Find m, number of sensing locations

Find r, data rate of the sensor selected

Find m', number of sensors per transducer

$m' = \text{floor} (BW_{max}/r)$

While ($m' \times PBR \times 2 >= BUFmax$)

$m' = m' - 1$;

Find T, number of transducers required

$T = \text{ceil} (m'/m)$

$TL_{buses} = 0$ (Initialization)

(Contd.)

Determination of number of Transducers & their positions (Contd):

For every transducer, t_i in T {

Determine the position & sensor list of t_i

Find $\sum Lst_i$, Length of buses used to connect all sensors in the list to t_i

$TL_{buses} = TL_{buses} + \sum Lst_i$

}

$WST[j] = m^* WST[j] + TL_{buses} + T^*WT_{min}$

Store position [j] //position of transducer, t_i

}

WT [k] = min (WST [j], $\forall$ j)

Position [k] = position of transducer that give least weight for this sensing type

}

Figure 12: Sample Weight Heuristic to select transducers

NOTE: Weight heuristic principle: Select a sensor of the sum of its weight and the weights of the lightest instances of the other components satisfies the entire system's weight constraint. This principle is applied for every component.

References

[1] Kathail, Vinod, et al., Sept 2002, PICO: Automatically Designing Custom Computers, Computer, pp.39-47.

[2] Tanenbaum, Andrew S., 1999, Computer Networks, Prentice Hall, 3 Ed.

[3] Willis, R.R. and E.P. Jensen, 1979, Computer-aided design of Software Systems, Proc. Of 4th Intl Conference of Software Engineering, Munich, Germany, pp. 116-125.

[4] Allen, Robert J., 1997, A Formal Approach to Software Architecture, Technical Report: CMU-CS-97-144, CMU.

[5] Mukkamala, Ravi, et al., 2001, Design and Analysis of a Scalable Kernel for Health Management of Aerospace Structures, 20th Digital Avionics Systems Conference, FL, USA.

[6] Barbacci, Mario R., et al., May 1998, Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis, Technical Report: CMU/SEI-97-TR-029, CMU.

[7] Kazman, Rick, et al., Aug 1998, The Architecture Tradeoff Analysis Method, Proc Of 4th Intl Conference on Engineering of Complex Computer Systems, Monterey, CA

[8] Klein, Mark and Rick Kazman, Oct 1999, Attribute-Based Architectural Styles, Technical Report: CMU/SEI-99-TR-022, CMU.

[9] Kazman, Rick, et al., May 1994, SAAM: A Method for Analyzing the Properties of Software Architecture, Proc of Intl. Conference on Software Engineering, Sorrento, Italy, pp. 81-90.

[10] Lebold, Mitchell, et al., 2002, A Framework for Next Generation Machinery Monitoring and Diagnostics, Proc of 56th Machinery Failure Prevention Technology, pp. 115-126.

[11] Thurston, Michael & Mitchell Lebold, 2001, Standards Developments for Condition-based Maintenance Systems, Proc of 55th Machinery Failure Prevention Technology, pp. 363-374.

[12] Navinchandra, D., 1991, Exploration and Innovation in Design, Springer-Verlag.

[13] -, OMG Unified Modeling Language, http://www.omg.org/.

[14] Pressman, Roger S., 1996, Software Engineering: A Practitioner's Approach, McGraw-Hill Publishers, 4th Edition.

[15] Garlan, David and Mary Shaw, Jan 1994, An Introduction to Software Architecture, Technical Report: CMU-CS-94-166, CMU.

[16] Patterson, David and John Hennessy, 1996, Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, 2nd Ed.

[17] Coulouris, George, et al., 2001, Distributed Systems: Concepts and Design, Addison-Wesley, 3rd Ed.