International Journal of Mechanical, Industrial and Aerospace Sciences
ISSN: 2517-9950
Vol:2, No:9, 2008

Abstraction Hierarchies for Engineering Design

Esra E. Aleisa and Li Lin

***Abstract*—**Complex engineering design problems consist of numerous factors of varying criticalities. Considering fundamental features of design and inferior details alike will result in an extensive waste of time and effort. Design parameters should be introduced gradually as appropriate based on their significance relevant to the problem context. This motivates the representation of design parameters at multiple levels of an abstraction hierarchy. However, developing abstraction hierarchies is an area that is not well understood. Our research proposes a novel hierarchical abstraction methodology to plan effective engineering designs and processes. It provides a theoretically sound foundation to represent, abstract and stratify engineering design parameters and tasks according to causality and criticality. The methodology creates abstraction hierarchies in a recursive and bottom-up approach that guarantees no backtracking across any of the abstraction levels. The methodology consists of three main phases, representation, abstraction, and layering to multiple hierarchical levels. The effectiveness of the developed methodology is demonstrated by a design problem.

***Keywords*—**Hierarchies, Abstraction, Loop-free, Engineering Design

I. Introduction

DESIGN Abstraction Hierarchies (DAHs) are used commonly to represent various large-scale and complex problems[1, 2]. Their values have been realized across a wide spectrum of applications mainly to reduce the complexity of problems and to improve solution efficiency [3]. DAHs are also used to speed up the development time, save resources, and provide aggregate intelligent output[4]. In addition, DAH produces designs that are easier to interpret validate and update compared to not using hierarchies. Moreover, DAHs help explore design alternatives and produce intelligent decisions at an early stage of the design or plan [5-7]. Furthermore, DHA assist in focusing on important aspects of the design problem [8, 9]. For computational efficiency, DAHs have also allows parallel execution of models [10], facilitates the utilization of the off-shelf models legacy [11], and enhances model reusability and rapid prototyping [12-16]. However, despite DAHs' significant benefits, there is a lack of formal methodologies for hierarchical abstraction generation suitable for design. In fact, hierarchical abstraction in general has been described as a "black art" [17]. In this research we aim to provide a formal hierarchical abstraction methodology to represent and plan engineering design problems at multiple levels of abstraction. Such that partial design solutions obtained at some abstraction level is preserved while the design is augmented at more detailed levels. The objectives of the methodology are three fold:

1. to develop a representation for engineering design that supports hierarchical abstraction,
2. to specify the clustering criteria according to which the abstraction process is defined, and
3. to develop a layering method, by which clusters of abstracted design parameters should be stratified in a hierarchy, without inducing any backtracking in the design process.

The reminder of this paper is structured as follows: first we provide a brief literature review of some of the most persistent abstraction systems and the reason why they are cumbersome when applied to engineering designs. This necessitates the need for this research. Next we dedicate a separate section to explain each of the three developmental phases of our hierarchical abstraction methodology. Then we provide some analysis on the methodology and theoretically proof that it is loop-free. Finally, we demonstrate the effectiveness of the methodology on the design process of a chemical processing system.

II. Literature Review

Although the nature of research on abstraction hierarchies is broad and multi-disciplinary, the most detailed work and thorough analysis of abstraction was conducted in the field of Artificial Intelligence (AI) [18, 19]. Abstraction models and systems were classified in various research efforts such as in [4, 6, 8, 11, 18, 20-24]. One of the earliest semi-automatic abstraction systems was ABSTRIPS [5, 25]. Based on a STRIPS (Stanford University Research Institute Planning System) framework, ABSTRIPS uses a state-space representation to create abstraction hierarchies by removing symbols from the formal language [22]. The successors of ABSTRIPS are many, including PRODIGY/EBL [26], ABTWEAK [27], PABLO [28], ALPINE [17], HIGHPOINT [29], STAR [22] and HW[19]. Other extensions incorporate the probabilistic distribution of the operators effects and a

Manuscript received October 15, 2008

E. A. Author is with the Department of Industrial and Management Systems at Kuwait University, Kuwait. (phone: 965-498-5249; fax: 965-481-6137; e-mail: aleisa@kuniv.edu).

distribution of probabilities on the possible initial states of a certain domain [30, 31], which incorporates the probabilistic distribution of the operators effects and a distribution of probabilities on the possible initial states of a certain domain. The main contribution of AI-based abstraction systems was in identifying properties that would render a hierarchical abstraction methodology effective. These include characterizing abstraction hierarchical methodology to be formal, complete, computable, produces simpler models, tractable and inexpensive to develop. Research efforts such as in [17, 32]. Bacchus and Yang [29, 33] have established properties that guarantee the effectiveness of abstraction methodologies. The essence of these properties is to maintain the structure of the solution that is obtained at more abstract levels while refining the solution quality at more detailed levels. The Downward Refinement Property (DRP) [33, 34]and the Ordered Monotonicity Property (OMP) [17] are two examples of such. Both properties have the advantage of being computable, tractable and capture a large spectrum of abstraction models [35]. However, these properties are heuristics that cannot guarantee a significant reduction in search space.

Despite their valuable contributions in characterizing effective abstraction practices, AI-based abstraction systems do not offer a convenient tool to construct abstraction hierarchies suitable for engineering design. That is because most of these systems are based on a STRIPS framework. The nature of primitive elements in the STRIPS language is cumbersome when used to describe engineering design. In fact, in some cases STRIPS representation can result in combinatorial issues [36]. Furthermore, most of AI-based abstraction systems require a goal state to be identified a priori which, presents a significant challenge for design problems. This is due to the fact that a designer might not be aware in advance what will be the final features of the design. For these reasons, there is an urgent need to develop hierarchical abstraction methodologies that utilize the AI-based abstraction advances but is tailored to the engineering design representation and requirements. For that we propose hierarchical abstraction methodology that consists of three phases: the representation phase, the abstraction phase, and finally the development of a design hierarchy. The details of the methodology are explained in the following sections.

III. Methodology for Generating Abstraction Hierarchies for Engineering Design

The developed abstraction methodology is based on the belief that details of a given design problem are not of equal importance. Design details need to be considered in sequent relative to one another for effective design planning. Failing to consider some precedence requirements when solving a design problem will result in resolving large parts of the problem if not the entire problem. This obviously will waste time and effort.

The methodology prescribes a partial order of design parameters under consideration, in a hierarchical representation. Such that no backtracking (looping) occurs throughout the design process. Eliminating backtracking implies that the structure of partial design solutions obtained at abstract levels need not be altered as more design details are introduced gradually, while the design process is evolving. The developed hierarchical abstraction methodology is depicted in Fig. 1.

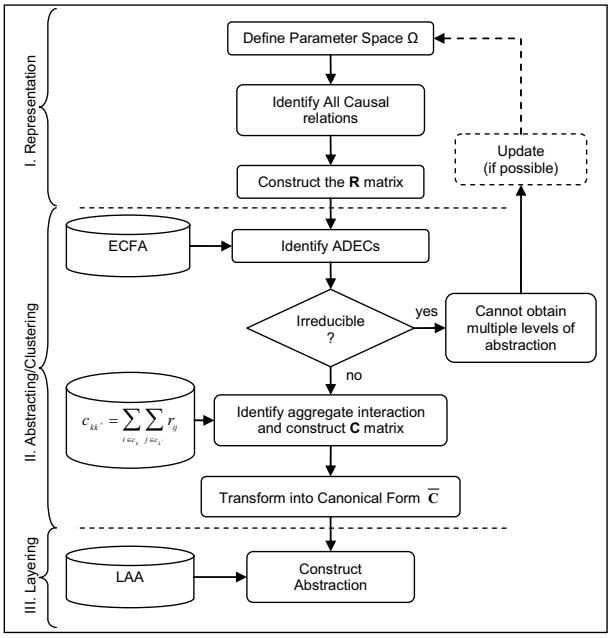

Fig. 1 Hierarchical abstraction methodology for design

As shown in Fig. 1, the methodology of developing abstraction hierarchies for engineering design consists of three phases:

1. Representation,
2. Abstraction/clustering, and
3. Layering.

In the representation phase, the design parameters' space, denoted by **Ω**, is represented in a manner that would support the abstraction process [37]. This is accomplished by identifying the causal relationships between the different design parameters of **Ω** which are represented by what is so called the **R** matrix.

In the abstraction phase, design parameters are clustered into their abstract design equivalence classes (ADECs) using

an equivalence class formation algorithm (ECFA) and the interaction matrix **R** as an input. In this phase, if the parameter space Ω is found to be irreducible, i.e., belongs to a single equivalence class, then, we conclude that all the design parameters of Ω communicate with one another. This means that all the design parameters need to be considered simultaneously. If this is the case, then we conclude that using a DAH will add no benefit to the original problem representation. The analysts can choose not to consider hierarchical abstraction or revise the problem definition to eliminate some of the interactions causing irreducibility. However, in some cases this may sacrifice the problem integrity or even might not be possible, due to the criticality of some interactions among some design parameters. For that reason this part is illustrated by dashed lines in Fig. 1. On the other hand, if Ω is *not* irreducible, the aggregate flows or interactions among ADECs are calculated (using Eq.(10)) and the aggregate flow matrix denoted by **C** is constructed accordingly. Then the **C** matrix is transformed into its canonical form written as $\overline{\mathbf{C}}$ to prepare it for the layering phase.

Finally, in the layering phase, all the ADECs of Ω are assigned to their appropriate abstraction level in a hierarchy in a way that would eliminate any backtracking or looping. The assignment of ADECs to the different hierarchical levels is accomplished using a level assignment algorithm (LAA). The details of the three methodology phases are explained in the following sections.

IV. Phase I: Representation of Engineering Design for Abstraction

To achieve efficiency in abstraction, the engineering design representation should support the clustering criteria according to which the abstraction is defined. Since our aim is to develop abstraction hierarchies that eliminate backtracking, the representation scheme should focus on causal relationships among the different design parameters. Therefore, we will use a parametric design representation that highlights the causal relationships between the design parameters under consideration.

A. The Parametric Representation of Design

A parameter design space Ω is a finite (countable) space of all design parameters under consideration. We use p_i to denote parameter i that belongs to Ω, such that $\mathrm{W} = \{p_1, p_2, \ldots, p_n\}$ or $|\mathrm{W}| = n$.. If the determination of design parameter p_i affects the value of the design parameter p_j, we say that that parameter p_i accesses parameter p_j, through causal link r_{ij}. Therefore, r_{ij} denotes the weight or the extent of causality from p_i to p_j, such that,

$$r_{ij} = \begin{cases} > 0 & p_i \text{ affects } p_j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

To maintain the direction of causal link r_{ij}, we restrict it to be nonnegative, i.e., $r_{ij} \geq 0$ for all i and j.

Since we can define r_{ij} between every pair of parameters in Ω, it is convenient to represent these weights in a two dimensional matrix $\mathbf{R}_{(n \times n)}$. To be able to trace indirect accessibility (or simply accessibility) we can use matrix multiplication. Let $\mathbf{R}^{(s)}$ denote that matrix **R** is multiplied s times by itself. Based on matrix theory we can interpret $r_{ij}^{(s)} > 0$ as the ability to reach p_j from p_i passing through s causal links (interactions). This is shown by Theorem 1 provided below. The proof for Theorem 1 provided in the appendix.

Theorem 1: Interpretation of $r_{ij}^{(s)} > 0$

If $r_{ij}^{(s)} > 0$ for some $s > 0$, then p_j is accessible from p_i by passing through s interactions(causal links).

The theorem leads to the definition of accessibility and communication provided next.

B. Accessibility between Design Parameters

Definition IV.1: Parameter accessibility

Let $p_i, p_j \in \mathrm{W}$, p_j is accessible from p_i ($accessible(p_i, p_j)$) if and only if $\exists r_{ij}^{(s)} > 0$ for a some $s = 1, 2, \ldots$.

In this research we refer to $r_{ij}^{(1)}$ by r_{ij} for simplicity. Also, it is reasonable to assume that each parameter affects itself, so we state that every parameter is at least accessible by itself, that is:

$$r_{ij} > 0 \ , \forall i = j \quad (2)$$

Moreover, accessibility is transitive, since:

$$\forall p_i, p_j, p_k \in \mathrm{W}, \begin{matrix} accessible(p_i, p_j) \cap accessible(p_j, p_k) \\ \Rightarrow accessible(p_i, p_k) \end{matrix} \quad (3)$$

Reflexiveness and transitivity makes accessibility a weak ordering relation [38] that can have a partial ordering relation induced onto it. This has the significance of enabling partial ordering for the parameters of W, which is the basis for our developed abstraction methodology. When two parameters are accessible to each other, we say that they communicate. Communication is defined below.

C. Communication among Design Parameters

Definition IV.2: Parameter communication

Let $p_i, p_j \in \mathrm{W}$, p_i and p_j communicate ($communicate(p_i, p_j)$) if and only if the following holds:

$$accessible(p_i, p_j) \cap accessible(p_j, p_i) \quad (4)$$

Alternatively, we can say that $p_i, p_j \in W$ communicate if there exists $r_{ij}^{(s)} > 0$ and $r_{ji}^{(s)} > 0$ for some $s = 1,2,\ldots$. Communication has the following properties:

1. $communicate(p_i, p_i)$, $\forall p_i \in W$ (5)

2. $communicate(p_i, p_j) \Leftrightarrow communicate(p_j, p_i)$, $\forall p_i, p_j \in W$ (6)

3. $communicate(p_i, p_j) \wedge communicate(p_j, p_k) \Rightarrow communicate(p_i, p_k)$, $\forall p_i, p_j, p_k \in W$ (7)

Equation (5) indicates that communication is reflexive, which is legitimate due to the reflexiveness of accessibility. Equation(6) shows that communication is symmetric, which is true by definition. Moreover, Eq.(7) points to the transitivity of communication that is directly deduced from applying Eq.(3) to the communication definition.

A relation that is reflexive, transitive and symmetric such as communication is an equivalence relation[38]. According to [39], an equivalence relation has the ability to partition the problem space upon which it is defined to disjoint partitions. We will use the communication partitioning ability to cluster communicating group of parameters into abstract equivalence classes. This will be achieved in phase II of the methodology explained in the next section.

V. Phase II: Clustering to Abstract Design Classes

We construct the abstract design space by clustering related design parameters into their abstract design equivalence classes (ADECs) according to specified clustering criteria. In this research, we utilize communication relations as the criteria to cluster design parameters into ADECs. ADECs are formally defined below.

Definition V.1: Abstract design equivalence class (ADEC)
An ADEC denoted by $c_k \subseteq W$, $k = 1,2\ldots$ is a *set* of design parameters, by which all the members belonging to it communicate with one another.

Hence determining the value of a design parameter affects the values of all other design parameters that are members of the same class. Moreover, because the clustering is based on an equivalence relation, that is communication, the fowling must hold for all ADECs:

1. $\bigcap_{\forall k} c_k = \emptyset, \forall k$ (8)
2. $\bigcup_{\forall k} c_k = W, \forall k$ (9)

Definition V.2: Irreducible parameter space
The parameter space W is said to be *irreducible* if: $\exists c_k$ such that $c_k = W$.

Irreducibility implies that the entire parameter space communicates with one another, hence belongs to a *single* ADEC. We will later show that there will be no gain when applying the developed abstraction methodology to domains with an irreducible parameter space.

A. Algorithm for Clustering Design Parameters into ADECs

In this section we explain an Equivalence Class Formation Algorithm (ECFA) that identifies communicating design parameters, and clusters them into their subsequent, disjointed ADECs. Developed by Gaver and Thompson [40][1], ECFA identifies ADECs by calculating *to-lists* and *from-lists*. The *to-lists* of parameter i, denoted by T_i, contains all the parameter that p_i can access in one or more steps. Similarly a *from-list* of p_i called F_i contains all the parameters from which p_i is accessible in one or more steps. Gaver and Thompson [40] showed that an equivalence class containing p_i denoted by c_i is the intersection of the sets T_i and F_i :

$$c_i = T_i \cap F_i, \forall i \quad (10)$$

B. Aggregate Interactions Among ADECs

The classification of design parameters into ADECs leads to the discussion on aggregate interactions or flows that result among them. In previous sections, we used matrices to represent the interactions among parameters; we intend to carry on the same process for the aggregate interactions.

Definition V.3: ADEC Interaction matrix
Let c_k and $c_{k'}$ be two ADECs. The interaction matrix $\mathbf{C}$ is a two dimensional matrix such that each entry $c_{kk'}$ of $\mathbf{C}$ is defined as follows:

$$c_{kk'} = \sum_{i \in c_k} \sum_{j \in c_{k'}} r_{ij} \quad (11)$$

$\mathbf{C}$ is a square matrix of size $m \times m$, where m is the number of ADECs in Ω. Each $c_{kk'}$ represents the amount of aggregate interactions that exists among the subsequent parameters of the two ADECs k and k', which are mathematically the summation of corresponding rows and columns of the $\mathbf{R}$ matrix. We use $\mathbf{C}^{(s)}$ to denote the $\mathbf{C}$ matrix multiplied s times by itself. Based on the proof of Theorem 1 given in the appendix (we did not include a separate theorem for $c_{kk'}^{(s)} > 0$ to avoid repetition), we can

[1] ECFA algorithm was originally used to obtain the communication classes of different states in the state-space of Markov chains. We shall modify the description of that algorithm to what best suits the problem representation of design parameter space.

easily show that if $c_{kk\not\in}^{(s)} > 0$ for some $s = 1,2,\ldots$, then there is an interaction between the two ADECs k and $k\not\in$ passing through s aggregate interactions. Hence we say that ADEC $k\not\in$ is accessible from ADEC k. This leads to the definition of ADEC accessibility.

Definition V.4: Accessibility of ADECs
If $c_k, c_{k\not\in} \in W$ are two ADECs, then we say that $c_{k\not\in}$ is accessible from c_k ($classaccessible(c_k, c_{k\not\in})$) if and only if there exists:

$$c_{kk\not\in}^{(s)} > 0 \text{ , for some } s = 1,2,\ldots \tag{12}$$

As for parameter accessibility, class accessibility has the following properties:

1. Reflexive, since:

$$classaccessible(c_k, c_{k\not\in}), \forall k = k\not\in \tag{13}$$

2. Transitive, due to:

$$classaccessible(c_k, c_{k\not\in}) \wedge classaccessible(c_{k\not\in}, c_{k\not\in\not\in}) \Rightarrow classaccessible(c_k, c_{k\not\in\not\in}) \tag{14}$$
$$\forall k, k\not\in, k\not\in\not\in$$

As indicated earlier a relation that exhibits reflexiveness and transitivity is a weak ordering relationship [38]. This property will be used later to partially order the design parameters of the design space Ω in a DAH. The details of this process will be explained in the analysis section of this paper.

Another important characterization of ADECs is whether a class is absorbing or transient. The distinction between these types of classes is provided in the following definitions.

Definition V.5: Absorbing ADEC (AADEC)
An AADEC $c_k \in W$ is one where:

$$c_{kk\not\in} = 0 \text{ , } \forall k \neq k\not\in \tag{15}$$

Definition V.6: Transient ADEC (TADEC)
A TADEC $c_k \in W$ is one where:

$$\exists\, c_{kk\not\in} > 0 \text{ , } k \neq k\not\in \tag{16}$$

In other words, an AADEC is a class that does not access any class other than itself. However, a TADEC as one that is able to access other classes beside itself.

C. Canonical Form of the C Matrix

To prepare the $\mathbf{C}$ matrix for the layering phase we rearrange its rows and columns, such that the first $m - t$ ones contain the AADECs, while the remaining t ones contain the TADECs. When this segregation is applied to the $\mathbf{C}$ matrix, then it is said to be in the canonical form denote it by $\bar{\mathbf{C}}$. A general structure of a $\bar{\mathbf{C}}$ matrix is given in the matrix below:

$$\bar{\mathbf{C}} = \begin{array}{c} \\ m-t \\ t \end{array} \overset{\begin{array}{cc} m-t & t \end{array}}{\left(\begin{array}{cc} \mathbf{D} & \mathbf{0} \\ \mathbf{T} & \mathbf{Q} \end{array}\right)}$$

The resultant submatrices of $\bar{\mathbf{C}}$ are as follows:

1. $\mathbf{D}_{(m-t)\times(m-t)}$ is a diagonal matrix, because it depicts the interaction among AADECs only. Note that an AADEC has access to no other class but itself (see Eq.(15)).
2. $\mathbf{0}_{(m-t)\times(t)}$ consists entirely of zeros, since it is not possible to have interaction from AADECs to TADECs.
3. $\mathbf{T}_{(t)\times(m-t)}$ depicts the interactions from each TADEC to each AADEC.
4. $\mathbf{Q}_{(t)\times(t)}$ depicts the interactions among all the TADECs of Ω.

The canonical form has been used requisitely in the literature of Markov Chains to study the behavior of the chain until absorption. Here the canonical form will be used as an effective tool to organize the aggregate interactions between different ADECs. This will shortly be used to assign each ADEC to its appropriate level of abstraction in a DAH.

VI. Phase III: Layering ADECs to Multiple Hierarchical Levels

Having defined the relational properties that characterize different ADECs, we are now ready to assign each ADEC to its appropriate level of abstraction in a DAH. The layering process is designed to eliminate any backtracking in the design plan.

The construction of a DAH is conducted in a recursive and bottom-up manner. It starts building the hierarchy from the lowest level of detail (level zero) and subsequently builds higher levels based on the abstract class accessibility relationships that exist among different ADECs.

Level zero is designated to include the details that can be postponed until the end when solving the problem hierarchically. However, level n, the highest level of abstraction, includes the details that need to be considered in the beginning. Therefore, the algorithm executes the hierarchy in a *last in first out* (LIFO) basis, as it builds the hierarchy in a

bottom-up fashion, but expects it to be executed in a top-down fashion (see Fig. 2).

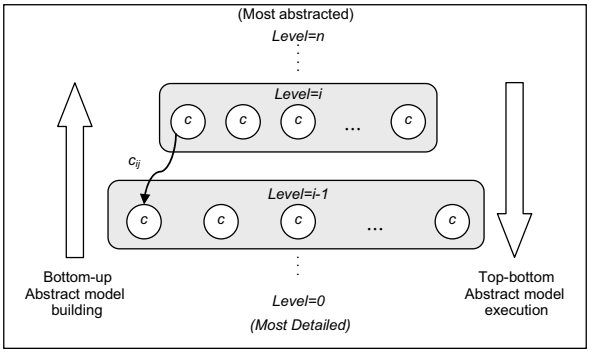

Fig. 2 Bottom-up abstract model building and top-bottom execution

The assignment of design parameters to levels is based on the theorems given below. The proof for each of these theorems is provided in the appendix.

Theorem 2*: The assignment of design parameter to abstraction levels*

Let $level(p_k)$ denote the level of the design parameter p_k in a DAH. For all $p_i, p_j \in W$, if $r_{ij}^{(s)} > 0$ for some $s > 0$, then we must have $level(p_i) \geq level(p_j)$ to avoid backtracking.

The above theorem indicates that if p_i affects p_j, then p_i should at least be at the same or a higher level than p_j.

Theorem 3*: The assignment of communicating design parameters to abstraction levels*

Let $level(p_k)$ denote the level of design parameter p_k in the DAH. For all $p_i, p_j \in W$, if $communicate(p_i, p_j)$, then $level(p_i) = level(p_j)$.

Theorem 3 indicates that communicating parameters are equivalent in terms of level. Therefore, they need to be assigned the same level to avoid backtracking.

Theorem 4: The assignment of ADECs to abstraction levels

Let $level(c_k)$ denote the level of ADEC k in a DAH. For all $c_i, c_j \in W$ where $i \neq j$, if $c_{ij} > 0$, then $level(c_i) > level(c_j)$ to avoid backtracking.

Based on the above theorems, we can conclude that considering ADECs (not parameters) is sufficient to produce a DAH with no backtracking. Next, we incorporate the above theorems into a level assignment algorithm that is able to generate automatic loop-free DAHs.

A. The Level Assignment Algorithm

The Level Assignment Algorithm (LAA) will generate DAHs by assigning ADECs to their appropriate level of abstraction. LAA works on the premises of the level assignment theorems provided in the previous section. The input for LAA is an **Ω** that is *not* irreducible. If **Ω** was irreducible, then based on the definition of irreducibility, the design parameters of the entire design space would communicate with one another. In that case, according to Theorem 3, all the ADECs and their subsequent parameters are assigned to the same level. Therefore, we can conclude that a hierarchy cannot be generated, as it would be pointless to have a hierarchy of a single abstraction level.

As shown in

Fig. **3**, the LAA first initializes the *level* variable to zero and the ADEC index i to one. Moreover, initially the *unassigned array* will contain all the ADECs while the *assigned array* is empty. LAA first assigns all AADECs to the lowest level of detail, and then assigns TADECs to higher levels of abstraction, according to the ADEC accessibility relationship. The details of these assignments are discussed below.

1) The Assignment of AADECs

By definition, an AADEC does not access to classes other than itself. Therefore, the design parameters that belong to an AADEC do not affect any parameters of other ADECs. For that reason, they can be considered as late as possible while solving the problem. Therefore, LAA places them in the lowest level of the DAH, that is, level zero. This complies with the establishment of Theorem 1. The assignment of AADECs is depicted in the shaded part of LAA in

Fig. **3**. In this part, the algorithm scans all c_k, $k = 1, 2, \ldots, m$ for AADECs. If c_k is absorbing, then, and c_k is removed from the *unassigned array* and added to the *assigned array*. The process repeats until no more AADECs are found. Then, the level is raised by one.

2) The Assignment of TADECs

After assigning all AADECs to level zero, LAA scans the **C** matrix for any classes in the *unassigned array* that access any of the classes in the *assigned array* and assigns them to the current level. This assignment is based on Theorem 4.

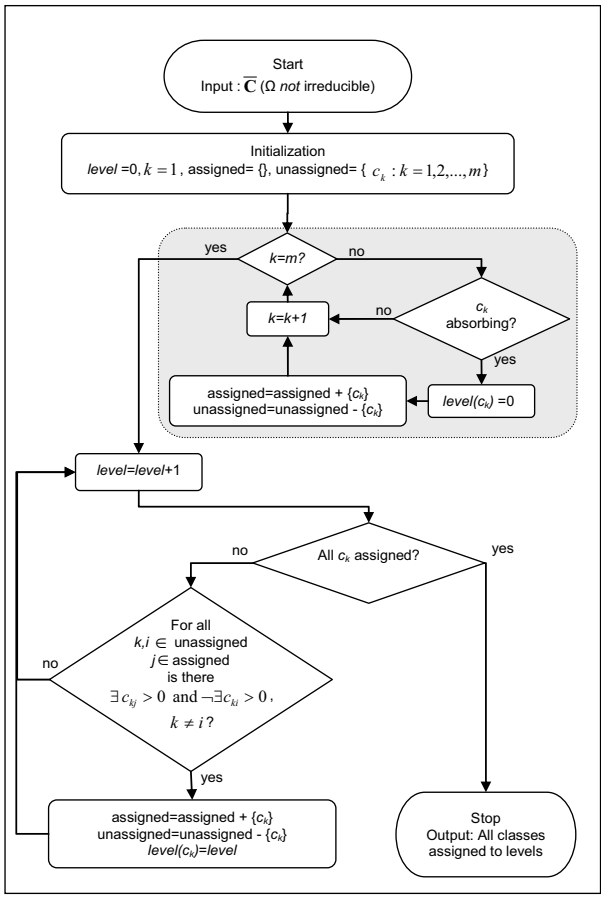

Fig. 3 The level assignment algorithm

In particular, due to accessibility, if $c_{ij} > 0$ then c_i contains at least one design parameter that affects some parameter in c_j. This means that c_i needs to be considered before c_j and is hence placed in a higher abstraction level ($level(c_i) > level(c_j)$). However, we need to make sure that there is no other TADEC c_k that is accessed by c_i, where, $c_{ik} > 0$. This is because we must have $level(c_i) > level(c_k)$, which is one level higher than the current level. In addition, we must have $level(c_j) > level(c_i) > level(c_k)$ to guarantee no backtracking (see Theorem 4). If this is the case ($\$c_{ik} > 0$), then we postpone the assignment of the level of c_i until an appropriate time, otherwise $level(c_i) = level$ which is currently equal to one. Accordingly, we update the *unassigned array* by removing c_i and adding it to the *assigned array*. The variable *level* is raised by one again. And the process repeats until all TADECs are in the *assigned array.*

The next section provides analysis to prove that any DAH produced using the developed methodology is loop-free.

VII. Analysis on the Hierarchical Abstraction Methodology

In the previous sections we presented an integrated recursive bottom-up methodology to build DAHs of a parameter space. The objective of the methodology was to produce loop-free abstractions. A loop-free abstraction is one where backtracking never occurs across the levels of the DAH, which is the key to the success of the developed methodology. In this section we will theoretically demonstrate that DAHs developed by the methodology in hand are loop-free.

Definition VII.1: Backtracking
Let c_i, c_j Ì Wbe two ADECs, a *backtrack* b occurs if:

1. $classaccessible(c_i, c_j)$ holds, and
2. $level(c_j) > level(c_i)$

Therefore, a backtrack b implies that c_j was solved before c_i, because $level(c_j) > level(c_i)$. However, as c_i affects c_j, due to $classaccessible(c_i, c_j)$, it will require resolving c_j, hence a backtrack.

Definition VII.2: Loop-Free
A DAH is *loop-free* if backtracking never occurs across any of its abstraction levels.

The following definitions will assist will assist in proving that the developed hierarchical abstraction methodology produces loop-free DAHs.

Definition VII.3: Weak ordering relation
A relation T that is reflexive and transitive is called a weak ordering relation.

Definition *VII.3* was obtained from the theory of ordered relations [39]. In previous sections we have shown that accessibility in general and class accessibility in particular are reflexive and transitive, and thus are weak ordering relations (WOR). The significance of WOR lies in their ability to order the design parameters of **Ω** according to the relation upon which it is defined. In our case, this relation is the accessibility relationship.

It is worthwhile to discuss the relation between parameter accessibility and class accessibility in terms of the ordered relations theory. This will help justify why class accessibility is sufficient enough to produce loop-free abstractions. This discussion will be based on the following definition, which is also obtained from the theory of ordered relations [39].

Definition VII.4: Induced relation

Let T be a WOR, and T^* is a relation defined on the equivalence classes of T so that uT^*v hold between two equivalence classes if and only if xTy, whenever $x \in u$ and $y \in v$, then T^* is an induced relation on T.

Based on Definition VII.4, class accessibility is a WOR induced on accessibility, because it is defined on the equivalence classes of accessibility. Specifically, $classaccessible(c_k, c_{k\neq})$ hold if and only if $\exists p_i \in c_k, p_j \in c_{k\neq}$, where $accessible(p_i, p_j)$ holds.

Definition VII.5: Partial ordering

A WOR whose equivalence relation is the identity relation is called a partial ordering.

We have shown earlier that communicating parameters are equivalent in terms of order (see Theorem 3) such that, if $communicate(p_i, p_j)$, then $level(p_i) = level(p_j)$. Since ADECs are constructed based on the communication relationship, then the induced class accessibility relation is a partial ordering (PO). A PO obtains its name due to the fact that the elements of each equivalence class are not ordered with respect to one another [41]. Note that PO is a special case of WOR as it has the property that none of its elements are alike. In other words, we cannot have an identity relation among ADECs. Therefore, the resultant DAH based on PO is loop-free (i.e. no loops among classes). This result was also indicated in Theorem 4, where if $c_{ij} > 0$, then we must have $level(c_i) > level(c_j)$ to avoid backtracking. We can also reach the same result by defining *minimal* and *maximal* elements.

Definition VII.6: Maximal and minimal elements

Let T be a WOR, an element $x \in W$ is called a *maximal* if $\nexists y \in W$, for which yTx. If x is unique then it is called a *maximum*. Similarly, x is a *minimal* element if $\nexists y \in W$ for which xTy holds, if x is unique then it is called a *minimum*.

Applying the above definition to class accessibility, we conclude that an AADEC is a *minimal*, since $c_{kk\neq} = 0$ for all $k \neq k\neq$. On the other hand, a TADEC c_k that is not accessed by an ADEC other than itself ($c_{k\neq} = 0$ for all $k \neq k\neq$) is a *maximal*.

Theorem 5: Loop-Free abstraction

Any DAH developed using class accessibility is loop-free

Proof

Looping (backtracking) occur if $\exists c_i, c_j \in W$, where $classaccessible(c_i, c_j)$ and $level(c_j) > level(c_i)$. We will show that this never occurs, considering the three cases of *minimal, maximal* and *intermediate* elements.

Case I (minimal): if c_i is absorbing, then, it is a *minimal* element, and $level(c_i) = 0$. Then $\nexists c_j \in W$, where $classaccessible(c_i, c_j)$, thus $level(c_j) > level(c_i)$ cannot occur.

Case II (maximal): if c_i is a *maximal* element, then $level(c_i) = n$. Thus $\nexists c_j \in W$, where $level(c_j) > level(c_i)$.

Case III (intermediate): if c_i is an element neither a *maximal* nor a *minimal*, then it must be true that $\exists c_j \in W$, where $classaccessible(c_j, c_i)$. And it also must be true that $\exists c_k \in W$, such that $classaccessible(c_i, c_k)$. Since class accessibility is a PO then $level(c_k) < level(c_i) < level(c_j)$ and a reverse order can never occur.

From these three cases, we conclude that $level(c_j) > level(c_i)$ will never occur for all $classaccessible(c_i, c_j)$. Hence it is loop-free.

By providing this proof, we have demonstrated that a DAH developed by the methodology in hand will always produce loop-free hierarchical abstractions. Now we will apply our methodology to an engineering design example.

VIII. An Illustrative Example

We now demonstrate the development of a DAH using the hierarchical abstraction methodology on an engineering design of a chemical processing system.

A. Phase I: Representation

As shown in Fig. 1, in the representation phase, we need to identify and represent the parameters for the design of the chemical processing system. Moreover, we need to represent interactions among all design parameters according to the **R** matrix representation.

1) Design Parameters of the Chemical Processing System

in [16, 42] provided a high level description of twenty design elements for a design of a chemical system. These elements represent the parameter space **Ω** and are shown in Table I.

TABLE I
DESIGN PARAMETERS OF THE CHEMICAL PROCESSING SYSTEM

Design Parameters	
p_1 : Operating structure design	p_{11} : Seismic design
p_2 : Vessel design	p_{12} : Piping design
p_3 : Plant layout/general arrangement	p_{13} : Process and instrumentation diagram
p_4 : Shipping design	p_{14} : Equipment support
p_5 : Structure lifting design	p_{15} : Pipe flexibility analysis
p_6 : Pressure drop analysis	p_{16} : Design documentation
p_7 : Process engineering	p_{17} : Foundation load design
p_8 : Structural documentation	p_{18} : Insulation structural design
p_9 : Size valves	p_{19} : Structural bill of material
p_{10} : Wind load design	p_{20} : Assembly design

2) The R Matrix for the Chemical Processing System

Based on the causal relations among the design parameters that were discussed in Chen and Lin [42], we can develop the corresponding **R** matrix using Eq. (1), where we use 1 to indicate a parameter accessibility between two design parameters, and zero otherwise. The **R** matrix is provided below:

$$\begin{array}{c} \\ \mathbf{R} = \end{array} \begin{array}{c} \begin{array}{cccccccccccccccccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} & p_{11} & p_{12} & p_{13} & p_{14} & p_{15} & p_{16} & p_{17} & p_{18} & p_{19} & p_{20} \end{array} \\ \begin{pmatrix} 1 & & & 1 & 1 & & & & & 1 & & & & 1 & & 1 & & & & \\ & 1 & & & & & & & & 1 & & & 1 & 1 & 1 & 1 & & 1 & & \\ & & 1 & & & & & 1 & 1 & & 1 & & & & 1 & 1 & & & & \\ 1 & & & 1 & & & & 1 & & & & & & & & & 1 & & & 1 \\ & & & 1 & 1 & & & 1 & & & 1 & & & & & 1 & & & 1 & \\ & & & & & 1 & & & & & & & & 1 & & & & & & \\ & 1 & & & 1 & & 1 & & & & 1 & & 1 & 1 & & & & 1 & & \\ 1 & & & 1 & & 1 & & 1 & & 1 & & & & & & & & 1 & & \\ & 1 & & & 1 & & & & 1 & & & & & & & & 1 & & & \\ & & & & 1 & & & & & 1 & & & & & & 1 & & & 1 & \\ 1 & & & & 1 & & & 1 & & & 1 & & & & & & & 1 & & \\ & & & 1 & & & & & 1 & & 1 & 1 & 1 & & 1 & & & & & 1 \\ 1 & & & & & & & & & & & & 1 & & & & 1 & & & \\ & & & & & & & & & & & & & 1 & & & & & & 1 \\ 1 & & & 1 & & 1 & & & & & & & & 1 & 1 & & & & & \\ & & & & & & & & & & & & & & & 1 & & & & \\ & & & 1 & 1 & 1 & & & & & & & & & & & 1 & & & \\ & & & & & & & 1 & & & 1 & & & & & 1 & & 1 & 1 & \\ 1 & & & & & & & & & 1 & & & & & & & 1 & & 1 & \\ & & & & & 1 & & & & & & & & & & & & & & 1 \end{pmatrix} \end{array}$$

The above **R** matrix is the transpose of the design matrix provided in Chen and Lin [42]. This is because the causal relations in Chen and Lin [42] were defined in reversed order compared to the way they are defined in this research.

B. Phase II: Clustering and Abstraction

In this phase we cluster communicating design parameters into their subsequent mutually exclusive ADECs and calculate the subsequent aggregate flow.

1) Clustering the Design Parameters into ADECs

The clustering process for the design parameters is accomplished using ECFA. The corresponding design parameters' *to-lists*, *from-lists* and the resultant ADECs obtained using ECFA are listed in Table II.

In Table II, the design parameters for the chemical processing system have been abstracted to ten distinct ADECs. These are $c_1 = \{1,4,5,8,10,11,17,18,19\}$, $c_2 = \{2\}$, $c_3 = \{3\}$, $c_4 = \{6,14,20\}$, $c_5 = \{7\}$, $c_6 = \{9\}$, $c_7 = \{12\}$, $c_8 = \{13\}$, $c_9 = \{15\}$ and $c_{10} = \{16\}$. The abstract design space includes two AADECs or minimal elements, these are c_4 and c_{10}, while the remaining are TADECs. Since we have more than a single ADEC, we can conclude that the design parameter space Ω is *not* irreducible. As discussed earlier, this provides an early indication that DAHs will be beneficial to the problem in hand. One distinctive benefit of applying DAHs to the design of the chemical processing system is the reduction of the problem size to half of its original parametric problem representation.

2) The C Matrix for the Chemical Processing System

Now we calculate the aggregate interactions among the AADECs and the TADECs of the chemical processing system using Eq.(10). The corresponding **C** matrix is provided below:

$$\begin{array}{c} \\ \mathbf{C} = \end{array} \begin{array}{c} \begin{array}{cccccccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} \end{array} \\ \begin{pmatrix} 34 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 4 \\ 2 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

The canonical form $\bar{\mathbf{C}}$ of the **C** segregates the AADECs of **C** in the first two rows and columns from the TADECs, which are transferred to the remaining eight rows and columns of $\bar{\mathbf{C}}$. $\bar{\mathbf{C}}$ matrix for the chemical processing system is:

$$\bar{\mathbf{C}} = \begin{array}{c} \begin{array}{cccccccccc} c_4 & c_{10} & c_1 & c_2 & c_3 & c_5 & c_6 & c_7 & c_8 & c_9 \end{array} \\ \left(\begin{array}{cc|cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 4 & 4 & 34 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 2 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) \end{array}$$

TABLE II
CREATING ADECS USING TO AND FROM LISTS

p_i	To–Lists	From–Lists	ADECs
p_1	{1, 4, 5, 10, 14, 16, 8, 17, 20, 11, 19, 6, 18}	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19	{1, 4, 5, 8, 10, 11, 17, 18, 19}
p_2	{2, 10, 13, 14, 15, 16, 18, 5, 19, 1, 17, 20, 4, 6, 8, 11}	2, 3, 7, 9, 12	{2}
p_3	{3, 8, 9, 11, 15, 16, 1, 4, 6, 10, 18, 2, 5, 17, 20, 14, 19, 13}	3	{3}
p_4	{4, 1, 8, 17, 20, 5, 10, 14, 16, 6, 18}	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19	{1, 4, 5, 8, 10, 11, 17, 18, 19}
p_5	{5, 4, 8, 11, 16, 19, 1, 17, 20, 6, 10, 18, 14}	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19	{1, 4, 5, 8, 10, 11, 17, 18, 19}
p_6	{6, 14, 20}	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20	{6, 14, 20}
p_7	{7, 2, 5, 11, 13, 14, 18, 10, 15, 16, 4, 8, 19, 1, 17, 20, 6}	7	{7}
p_8	{8, 1, 4, 6, 10, 18, 5, 14, 16, 19, 11, 17, 20}	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19	{1, 4, 5, 8, 10, 11, 17, 18, 19}
p_9	{9, 2, 5, 17, 20, 10, 13, 14, 15, 16, 18, 4, 8, 11, 19, 6, 1}	3, 9, 12	{9}
p_{10}	{10, 5, 16, 19, 4, 8, 11, 1, 17, 20, 6, 18, 14}	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19	{1, 4, 5, 8, 10, 11, 17, 18, 19}
p_{11}	{11, 1, 5, 8, 18, 4, 10, 14, 16, 19, 6, 17, 20}	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19	{1, 4, 5, 8, 10, 11, 17, 18, 19}
p_{12}	{12, 5, 9, 11, 13, 15, 20, 4, 8, 16, 19, 2, 17, 1, 18, 6, 14, 10}	12	{12}
p_{13}	{13, 1, 17, 4, 5, 10, 14, 16, 6, 8, 20, 11, 19, 18}	2, 3, 7, 9, 12, 13	{13}
p_{14}	{14, 20, 6}	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20	{6, 14, 20}
p_{15}	{15, 1, 4, 6, 14, 5, 10, 16, 8, 17, 20, 11, 19, 18}	2, 3, 7, 9, 12, 15	{15}
p_{16}	{16}	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19	{16}
p_{17}	{17, 4, 5, 6, 1, 8, 20, 11, 16, 19, 14, 10, 18}	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19	{1, 4, 5, 8, 10, 11, 17, 18, 19}
p_{18}	{18, 8, 11, 16, 19, 1, 4, 6, 10, 5, 17, 14, 20}	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19	{1, 4, 5, 8, 10, 11, 17, 18, 19}
p_{19}	{19, 1, 10, 17, 4, 5, 14, 16, 6, 8, 20, 11, 18}	1, 12, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19	{1, 4, 5, 8, 10, 11, 17, 18, 19}
p_{20}	{20, 6, 14}	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20	{6, 14, 20}

C. Phase III: Constructing the DAH for the Design Problem

In this phase, we utilize the interactions among the different ADECs to recursively develop a DAH for the design of the chemical processing system. As indicated in the methodology, DAHs are designed to be loop-free. In terms of the problem in hand, obtaining partial design solutions by determining the values of some design parameters obtained at a given abstraction level need *not* be altered as the design process progresses hierarchically to more detailed levels.

Each ADEC is assigned to its appropriate abstraction level using LAA. Here we shall illustrate a step-by-step application of LAA to the design of a chemical processing system.

1. *Initialization:* According to LAA, the DAH is constructed in a bottom-up manner, and executed top-down. Therefore, the first level to be created is level zero, which is the lowest level in the DAH. Hence, initially the variable *level* is bound to zero (*level=0*). Moreover, in the initialization step the *unassigned array* is set to contain all the classes while the *assigned array* is set to be empty.

2. *Level Assignment of AADECs:* Scanning the $\bar{\mathbf{C}}$, we will find that the submatrix $\mathbf{D}$ contains only two AADECs, which are c_4 and c_{10}. Thus, according to LAA they need to be assigned to *level* zero ($level(c_4) = level(c_{10}) = 0$). The *assigned array* and the *unassigned array* are both updated to include the recently assigned classes to the former and delete them from the latter ($assigned = \{c_4, c_{10}\}$, $unassigned = \{c_1, c_2, c_3, c_5, c_6, c_7, c_8, c_9\}$).When no more AADECs are found, the variable *level* is raised by one (*level=1*).

3. *Level Assignment of TADECs:* In this part of LAA, we iteratively assign each TADEC to its appropriate level of detail, according to two attributes : its relation to the recently assigned classes (currently depicted in the $\mathbf{T}$ submatrix of $\bar{\mathbf{C}}$), and the way TADECs interact with one another (illustrated in the $\mathbf{Q}$ submatrix of $\bar{\mathbf{C}}$). If an unassigned TADEC k accesses to an assigned class j ($c_{kj} > 0$) that belongs to the *assigned array,* but does *not* access any other unassigned class i ($\not\exists c_{ki} > 0, \forall i, k \neq i$) that belongs to the *unassigned array*, it is then assigned to the current level. The process repeats until all classes are assigned to some level in the DAH. Fig. 4 shows the iterations of LAA applied to the ADECs of the design problem of the chemical system.

4. *Output:* The output of the LAA is a complete specification of the level assignment of each ADEC in the design problem, which constitutes the DAH shown in Fig. 5. Fig. 1also illustrates the causal links among different levels of the DAH, where the solid lines are the ones between two consecutive levels and the dashed line depicts the ones otherwise. Moreover, the Fig shows no backtracking among any of the abstraction levels developed.

Iteration= 0 ,*level= 0*	*Unassigned cells (k,i)*									
(j)assigned={ }	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
Is *absorbing?*				x						x

Level(c_4)=0, Level(c_{10})=0

Iteration= 1 ,*level= 1*	*Unassigned cells (k,i)*							
(j)assigned={c_4,c_{10}}	c_1	c_2	c_3	c_5	c_6	c_7	c_8	c_9
Is $c_{kj}>0$?	x	x	x	x	x	x		x
Is *not* $c_{ki}>0$? ($k \neq i$)	x						x	
Assign to current level	x							

Level(c_1)=1

Iteration= 2 ,*level= 2*	*Unassigned cells (k,i)*						
(j)assigned= {c_1,c_4,c_{10}}	c_2	c_3	c_5	c_6	c_7	c_8	c_9
Is $c_{kj}>0$?	x	x	x	x	x	x	x
Is *not* $c_{ki}>0$? ($k \neq i$)						x	x
Assign to current level						x	x

Level(c_8)=2, Level(c_9)=2

Iteration= 3 ,*level= 3*	*Unassigned cells (k,i)*				
(j)assigned= {c_1,c_4,c_8,c_9,c_{10}}	c_2	c_3	c_5	c_6	c_7
Is $c_{kj}>0$?	x	x	x	x	x
Is *not* $c_{ki}>0$? ($k \neq i$)	x				
Assign to current level	x				

Level(c_2)=3

Iteration= 4 ,*level= 4*	*Unassigned cells (k,i)*			
(j)assigned= { c_1,c_2,c_4,c_8,c_9,c_{10}}	c_3	c_5	c_6	c_7
Is $c_{kj}>0$?	x	x	x	x
Is *not* $c_{ki}>0$? ($k \neq i$)		x	x	
Assign to current level		x	x	

Level(c_5)=4, Level(c_6)=4

Iteration= 5 ,*level= 5*	*Unassigned cells (k,i)*	
(j)assigned= { c_1,c_2,c_4,c_5,c_6 ,c_8,c_9,c_{10}}	c_3	c_7
Is $c_{kj}>0$?	x	x
Is *not* $c_{ki}>0$? ($k \neq i$)	x	x
Assign to current level	x	x

Level(c_3)=5, Level(c_7)=5
assigned= {c_1,c_2,c_3 ,c_4,c_5,c_6 ,c_7,c_8,c_9,c_{10}}
unassigned={}
STOP

Fig. 4 Application of the level assignment algorithm to the design of the chemical processing system

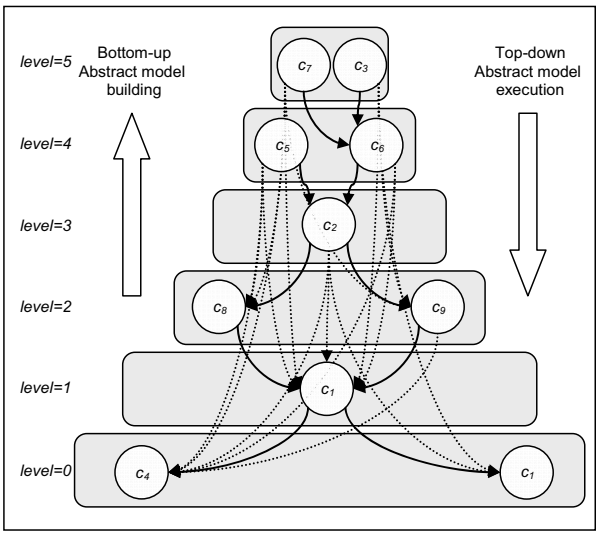

Fig. 5 Abstraction hierarchy for the design of the chemical processing system

IX. CONCLUSION

We have presented a hierarchical abstraction methodology suitable for engineering design. The developed hierarchical abstraction methodology consists of three phases: representation, abstraction, and layering of clustered abstract design parameters at multiple levels of the abstraction hierarchy. The methodology guarantees that partial design solutions obtained at higher levels of the hierarchy need not be altered as the design accrues gradually at lower detail levels. The developed abstraction hierarchies are recursively built bottom-up, but are executed top-down. A successful application of the methodology will facilitate improved decisions at early stages of the design, and allow the use of resources to focus on critical aspects of the design at its different phases. Moreover, the presented methodology identifies design tasks that are possible to accomplish concurrently. However, the extent of the gained efficiency largely depends on the context to which this methodology is applied. A design problem with an irreducible parameter space will result in an ineffective application, as the benefits of hierarchical representation will not be realized. Future research is directed towards identifying special cases of the parameter space that possess certain desirable characteristics, such as parallel execution. Moreover, further work will also examine the magnitude of causal relations to establish thresholds above which an interaction is considered significant enough to be accounted for when constructing abstraction hierarchies for engineering design.

X. APPENDIX

Theorem 1: Interpretation of $r_{ij}^{(s)} > 0$

If $r_{ij}^{(s)} > 0$ for some $s > 0$, then p_j is accessible from p_i by passing through s interactions(causal links).

Proof

Consider getting from p_i to p_j passing through two interactions. Then, there must be an intermediate parameter p_k to pass through to get to p_j, hence $r_{ij}^{(2)} = \sum_k r_{ik} r_{kj}$. This is the same as if we multiplied $\mathbf{R}$ by itself. Specifically $r_{ij}^{(2)}$ is the ij^{th} entry of the $\mathbf{R}^{(2)}$ matrix. Similarly, getting from p_i to p_j passing through three interactions. Then, $r_{ij}^{(3)} = \sum_k r_{ij}^{(2)} r_{kj}$. By matrix multiplication, $r_{ij}^{(3)}$ is the ij^{th} entry of the $\mathbf{R}^{(3)}$ matrix. Therefore, by mathematical induction $r_{ij}^{(s)}$ is the ij^{th} entry of the $\mathbf{R}^{(s)}$. Hence, $r_{ij}^{(s)} > 0$ indicates that we can reach p_j from p_i passing through s interactions.

Theorem 2: The assignment of design parameters to abstraction levels

Let $level(p_k)$ denote the level of the design parameter p_k in a DAH. For all p_i, p_j Î W, if $r_{ij}^{(s)} > 0$ for some $s > 0$, then we must have $level(p_i)$ ³ $level(p_j)$ to avoid backtracking.

Proof

The above theorem indicates that if p_j is accessible from p_i, then, p_i should be at that same or a higher level than p_j. By definition, if $r_{ij}^{(s)} > 0$ for some $s > 0$ holds, then design parameter p_i affects design parameter p_j. If we let $level(p_i) < level(p_j)$, then based on the top-bottom execution p_j will be solved before p_i. But p_i affects p_j, hence solving p_i requires *resolving* p_j. Since $level(p_i) < level(p_j)$, then this results in backtracking. Therefore, if $r_{ij} > 0$ for some $s > 0$, then $level(p_i)$ ³ $level(p_j)$ must hold to avoid backtracking.

Theorem 3: The assignment of communicating design parameters to abstraction level

Let $level(p_k)$ denote the level of design parameter p_k in the DAH. For all p_i, p_j Î W, if $communicate(p_i, p_j)$, then $level(p_i) = level(p_j)$.

Proof

If $communicate(p_i, p_j)$, then by definition there exists $r_{ij}^{(s_1)} > 0$ and $r_{ji}^{(s_2)} > 0$ for some $s_1, s_2 > 0$. Hence, by *Theorem* 2, $level(p_i)$ ³ $level(p_j)$ and $level(p_i)$ £ $level(p_j)$, which implies $level(p_i) = level(p_j)$.

Theorem 4: The assignment of ADECs to abstraction levels

Let $level(c_k)$ denote the level of ADEC k in a DAH. For all c_i, c_j Ì W where i ¹ j, if $c_{ij} > 0$, then $level(c_i) > level(c_j)$ to avoid backtracking.

Proof

The proof of this theorem is a direct result of applying Theorem 2 and Theorem 3. Theorem 4 indicates that if class c_i accesses c_j, then c_i should be placed at least one level higher than the level of c_j. Based on the definition of accessibility, if $c_{ij} > 0$ then, $\exists p_k$ Î c_i and $\exists p_l$ Î c_j such that $r_{kl}^{(s)} > 0$ for some $s > 0$. Based on Theorem 4, $level(p_k)$ ³ $level(p_l)$. Since classes consists of communicating parameters, then $level(c_i)$ ³ $level(c_j)$. But classes cannot communicate, then, it is not possible to have $level(c_i) = level(c_j)$ when $c_{ij} > 0$. Therefore, $level(c_i) > level(c_j)$ for $c_{ij} > 0$, and hence c_i need to be considered before c_j to avoid backtracking.

REFERENCES

[1] Lam, K.P., "Hierarchical Method for Large-Scale Two-Dimensional Layout". Journal of Mechanical Design, 1983. 105(2): p. 242-248.
[2] Sebastia, L., E. Onaindia, and E. Marzal, "Decomposition of planning problems". Ai Communications, 2006. 19(1): p. 49-81.
[3] Holte, R.C. and B.Y. Choueiry, "Abstraction and reformulation in artificial intelligence". Philosophical Transactions of the Royal Society of London Series B-Biological Sciences, 2003. 358(1435): p. 1197-1204.
[4] Goldin, S.E. and P. Klahr. Learning and Abstraction in Simulation. in International Joint Conference on Artificial Intelligence. 1981: American Assoc for Artificial Intelligence.
[5] Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces". Artificial Intelligence, 1974. 5(2): p. 115-135.
[6] Taylor, L.E. and M.R. Henderson. Roles of features and abstraction in mechanical design. in 6th International Conference on Design Theory and Methodology American Society of Mechanical Engineers, Design Engineering Division (Publication) DE. 1994. New York, NY: ASME.
[7] Reddy, S.Y., "Learning abstract models for system design". Ai Edam-Artificial Intelligence for Engineering Design Analysis and Manufacturing, 1996. 10(2): p. 167-169.
[8] Hoover, S.P. and J.R. Rinderle. Abstractions, design views and focusing. in 6th International Conference on Design Theory and Methodology American Society of Mechanical Engineers, Design Engineering Division (Publication) DE. 1994: ASME, New York, NY.
[9] Sarjoughian, H.S., B.P. Zeigler, and F.E. Cellier. Evaluating model abstractions: A quantitative approach. in Proceedings of SPIE Enabling Technology for Simulation Science II. 1998. Orlando, FL, United States.
[10] Kiran, A.S., T. Cetinkaya, and J. Cabrera, "Hierarchical modeling of a shipyard integrated with an external scheduling application". Winter Simulation Conference Proceedings, 2001. 2: p. 877-881 (IEEE cat n 01CH37304).
[11] McGraw, R.M. and R.A. MacDonald, "Abstract modeling for engineering and engagement level simulations". Winter Simulation Conference Proceedings, 2000. 1: p. 326-334.
[12] Zeigler, B.P., "Hierarchical, Modular Discrete-Event Modelling in an Object-Oriented Envirionment". Simulation, 1987. 49(5): p. 219-230.
[13] Lin, J.T., K.C. Yeh, and L.C. Sheu, "A context-based object-oriented application framework for discrete event simulation". Computers & Industrial Engineering, 1996. 30(4): p. 579-597.
[14] Praehofer, H., "Object oriented, modular hierarchical simulation modeling: towards reuse of simulation code". Simulation Practice & Theory, 1996. 4(4): p. 4.
[15] Pidd, M. and R.B. Castro. Hierarchical modular modelling in discrete simulation. in Winter Simulation Conference. 1998: IEEE, Piscataway, NJ.
[16] Chen, S.-J., "Project task coordination and team organization in concurrent engineering", in Department of Industrial Engineering. 1999, State University of New York at Buffalo: Buffalo, NY.
[17] Knoblock, C., "Automatically generating abstractions for planning". Artificial Intelligence, 1994. 68(2 Aug): p. 243-302.
[18] Giunchiglia, F. and T. Walsh, "A Theory of Abstraction". Artificial Intelligence, 1992. 57(2-3): p. 323-389.
[19] Armano, G., G. Cherchi, and E. Vargiu, "Planning by abstraction using HW[], in Book" in Planning by abstraction using HW[]. 2003. p. 349-361.
[20] Fishwick, P.A., "Role of Process Abstraction in Simulation". IEEE Transactions on Systems, Man & Cybernetics, 1988. 18(1): p. 18-39.
[21] Fishwick, P.A., Simulation model design and execution : building digital worlds. Prentice-Hall international series in industrial and systems engineering. 1995, Englewood Cliffs, N.J.: Prentice Hall. xvi, 448 p.
[22] Holte, R.C., et al., "Speeding up problem solving by abstraction: a graph oriented approach". Artificial Intelligence, 1996. 85(1-2 Aug): p. 321-361.

[23] Caughlin, D. and A.F. Sisti. A summary of model abstraction techniques. in Proceedings of SPIE - The International Society for Optical Engineering. 1997: SPIE.
[24] Sisti, A.F. and S.D. Farr. Model abstraction techniques: an intuitive overview. in National Aerospace and Electronics Conference, Proceedings of the IEEE 1998. 1998: IEEE, Piscataway, NJ.
[25] Giunchiglia, F., "Using Abstrips abstractions - Where do we stand?". Artificial Intelligence Review, 1999. 13(3): p. 201-213.
[26] Minton, S., "Learning Effective Search Control Knowledge: An Explanation-Based Approach". 1988, Carnegie-Mellon University. p. 231.
[27] Yang, Q. and J. Tenenberg. Abtweak: Abstracting a Nonlinear, Least Commitment Planner. in Proceedings of the 8th National Conference on Artificial Intelligence. 1990. Boston, MA.
[28] Christensen, J., "Automatic Abstraction in Planning", in Department of Computer Science. 1991, Stanford University: Stanford, Ca. p. 153.
[29] Bacchus, F. and Q. Yang. Expected value of hierarchical problem-solving. in AAAI-92. 1992.
[30] Friske, L.M. and C.H.C. Ribeiro, "Planning under uncertainty with abstraction hierarchies, in Book" in Planning under uncertainty with abstraction hierarchies. 2006. p. 1057-1066.
[31] Marie, d., R. Priyang, and G. Lise, "Learning structured Bayesian networks: combining abstraction hierarchies and tree-structured conditional probability tables". Computational Intelligence, 2008. 24(1): p. 1.
[32] Knoblock, C. Search reduction in hierarchical problem solving. in AAAI. 1991. Anaheim, CA.
[33] Bacchus, F. and Q. Yang, "Downward refinement and the efficiency of hierarchical problem solving". Artificial Intelligence, 1994. 71(1 Nov): p. 43-100.
[34] Helmert, M., "The Fast Downward planning system". Journal of Artificial Intelligence Research, 2006. 26: p. 191-246.
[35] Gimenez, O. and A. Jonsson, "The complexity of planning problems with simple causal graphs". Journal of Artificial Intelligence Research, 2008. 31: p. 319-351.
[36] Bylander, T., "Computational complexity of propositional STRIPS planning". Artificial Intelligence, 1994. 69(1-2): p. 165-204.
[37] Kemke, C. and E. Walker, "Planning with action abstraction and Plan Decomposition Hierarchies". 2006 Ieee/Wic/Acm International Conference on Intelligent Agent Technology, Proceedings, 2006: p. 447-451.
[38] Kemeny, J.G. and J.L. Snell, Finite markov chains. 1960, Princeton, N.J.,: Van Nostrand. 210 p.
[39] Dartmouth College Writing Group and E. Cogan, Modern mathematical methods and models; a book of experimental text materials. Vol. 2. 1958, Ann Arbor, MI.
[40] Gaver, D.P. and G.L. Thompson, Programming and probability models in operations research. 1973, Monterey, Ca: Brooks/Cole Pub. Co. xiii, 683 p.
[41] Russell, S.J. and P. Norvig, Artificial intelligence : a modern approach. Prentice Hall series in artificial intelligence. 1995, Englewood Cliffs, N.J.: Prentice Hall. xxviii, 932 p.
[42] Chen, S.J. and L. Lin, "A project task coordination model for team organization in concurrent engineering". Concurrent Engineering-Research and Applications, 2002. 10(3): p. 187-202.

Esra E. Aleisa is an assistant professor of Industrial and Management Systems Engineering (IMSE) at Kuwait University. She has received her Masters and PhD in Industrial Engineering and production systems from the department of Industrial Engineering at the State University of New York at Buffalo in 2001 and 2005 respectively. In 1998, she has earned her B.S. degree in industrial engineering from Kuwait University.

Dr. Esra Aleisa research interests include, Planning and design of large scale facilities, simulation and improvement of manufacturing and service systems, multilevel planning and design of complex engineering design, group technology (GT), and design structured matrices (DSM). She is a member of Omega Rho, the international operations research honor society, IEEE, INFORMS, IIE, ASEE.

LI LIN is Professor of Industrial Engineering at the University at Buffalo, the State University of New York. His research interests include concurrent engineering and product life cycle design, computer simulation, and manufacturing systems design and control. Dr. Lin s research has been supported by the National Science Foundation (NSF), the Environmental Protection Agency (EPA) and several industrial companies. He has published over forty papers in refereed research journals.