SYNTHESIS IN COMPLEX PROBLEM DOMAINS

INFORMATICA 4/89

Keywords: synthesis, complex systems, abstraction, hierarchical decomposition, structuring

Maksimilijan Gerkeš
Metalna Maribor

ABSTRACT:

Complex systems' synthesis becomes a bottleneck of production process. Some researchers denote this situation as crisis, others as "crisis". However, complex systems' synthesis especially those, which cannot be implemented within particular technology domain is an open problem.

The contribution gives a collection of procedures, developed with integration and some inovations of classical procedures and more advanced procedures. As a result, synthesis process can be based on abstraction, hierarchical decomposition, and structuring, while its derivation is nested in actual technological context, expressed through requests synthesized solution must satisfy.

POVZETEK: Sinteza v domeni kompleksnih problemov

Sinteza kompleksnih sistemov postaja vedno bolj ozko grlo produkcijskega procesa. Nekateri raziskovalci označujejo to stanje kot krizno, drugi kot "krizno". Kakorkoli, problem sinteze kompleksnih sistemov, posebej tistih, ki jih ni možno rešiti v domeni posamezne tehnologije, je dokaj odprt.

Prispevek podaja skupek postopkov izpeljanih, z integracijo in nekaj inovacijami, iz klasičnih postopkov in sodobnih postopkov sinteze. Kot rezultat lahko postopek sinteze gradimo na osnovi abstrakcije, hierarhične dekompozicije in strukturiranja, njegovo izvajanje pa je vgnezdeno v konkretni tehnološki kontekst izražen v obliki zahtev, ki jih mora izpolnjevati rešitev.

INTRODUCTION

Situation in high technology domains like CIM, flexible manufacturing, software, computer architectutes, VLSI, etc., seems similar regarding the requests for more efficient synthesis methods. For economic reasons synthesized solutions must be completely validated before manufacturing. A single error can cause exponential growth of operations to dispatch it. This calls for formal correct synthesis methods.

Complexity is common characteristic of systems in the above domains. It causes an enormous amount of operations, before the synthesis goal can be satisfied. Most of the contemporary synthesis methods seem to fail because of their too close technological orientation.

Rapid technology development disables efficient synthesis tools to be developed for each particular technology. Even if this is possible, there will still remain the problem of systems implemented in diverse technologies.

This clearly calls for synthesis methods, which will be technology independent and conceptualized on complexity and functionality issues of contemporary and advanced systems.

This contribution is based on system orientation to the synthesis problem, where current technology sets limits regarding system's functionality and structure. Such an approach allows that technology based descriptions are completely avoided until the solution representation level. Altough technology sets very exact limits during the synthesis process, technological objectives are not expressed explicitly, but reflect in

system structure and functionality.

An approach to synthesis is used, which is based on abstract system representation transformed through hierarchical decomposition and structuring to its counterpart, the solution. Abstraction and hierarchical decomposition have long been recognized as a means, which enables complexity reduction on one side and problem decomposition to several subproblems of lower complexity on the other side.

Structuring as a means to cope with complexity is more controversial and it seems there is no unisonous agreement about it.

However, combining abstraction and hierarchical decomposition in the synthesis process assures only that the initial and subsequent system structures are replicated and impressed in solution system structure. On the other hand it is well known that contemporary systems are supported with an enormous amount of software, which directs their actions and is a strict consequence of systems' structuring. To visualize this we can imagine that software through execution connects an object-flow graph that corresponds to problem structure and behaviour. This means that structuring is already extensively used in system design.

With respect to system represented with uncontrolled object-flow graph structuring requires three additional system structures. Its control structure directs subsystems , which can be at lowest representation level represented with uncontrolled object-flow graphs, connected in space and time. Particular strategy of control used is of secondary importance. Its memory structure assures object validity during controlled execution. Transfer structure delivers objects to subsystems' inputs. Both memory and transfer structures are under control of control structure. Those structures are usually partitioned throughout the system following the principles of distribution and hierarchy. More intensive structuring results in complex control scheme, which is usually but not necessary expressed in a form of control code or software. Making software more close to human comprehension capabilities does not change its nature. However, it is interesting that structuring does not necessary impose any control code or software.

This point of view allows that control code or software are determined as structuring side product. This is a conclusion, which sounds quite heretic, however, it gives at least theoretical possibility for automatic software development.

Basic structuring concept can be explained starting with a system with uncontrolled flow of objects. Observe that not all resources of such a system are active simultaneously. Assume a system in which resources are disconnected and a control mechanism, which is capable to connect them in time and space. Under certain conditions behaviour of both systems can be identical.

This simplified description, structuring is based on, should be completed with an observation that identical system resources can be shared when appropriate. To take advantage of this possibility system structure have to be completed with memory and transfer structures.

If the behaviour of a system have to change, its control functions which determine resource connections will be appropriately modified. The simplest way to do this is through program memory, which is a part of system's control structure.

Even when we look to software as problem description, system should be capable of its recognition and physical system restoration to solve the problem. A possible way to do this is that problem is represented with problems solvable with physical system resources. However, this introduce structure that have to be implemented with physical system in isomorphic way, directly or with appropriate time and space partition.

System synthesis can now be conceptualized based on the problem expressed in a form of abstract system, which have to be transformed through hierarchical decomposition and structuring to a solution, a system expressed with resources and structure that can be implemented in physical world.

Problem solvability seems to be in tight connection with its representation. Different representations can be used to express particular view on the problem. They are equivalent in the sense that they express the same behaviour in different ways. Synthesis procedures proposed are defined for each representation. Change of one representation to another is orderly developed.

1. OVERVIEW OF THE SYNTHESIS PROCESS

Problem specification consists of two parts. Problem definition part determines an abstract system for which the synthesis process have to determine a solution expressed in the form of physical system specification to be implemented in predetermined technology domain. Behaviour of physical system can be recognized as behaviour of abstract system merely through the use of suitable abstraction, otherwise no relation exists between the systems.

The rest of problem specification sets requests for the solution. Two classes, called the solution classes are determined based on the requests. Resource class consists of resource specifications, physical system should be built of. Resources determined with the resource class are generally composed and can be represented with resource structures of known behaviour. This usually allows resource class structuring through suitable relations. Abstraction through equivalence relation regarding resource behaviour reduces representation complexity and allows unnecessary details to be

neglected at earlier synthesis steps. More sophisticated abstraction procedures to reduce representational complexity are described in section 3.

Structure class consists of abstract resource structures. Physical system structure and its abstractions should be isomorphic to structures build of structures from structure class. Structure class is partially ordered with regard to substructure relation. Its representation complexity can be reduced through structure abstraction. Structure class can be empty. From the synthesis point of view this means that no structure requirements are set regarding the solution.

Intuitively, abstract and physical systems can be delimited with the notion of distance, which is a measure proportional to the number of synthesis steps required to reach the solution. Its numeric determination provides estimate about problem complexity - P, NP complexity as the roughest measure, for example. An argument will be given to show that the synthesis problem belongs to the domain of NP - complete problems, in general.

Initial synthesis steps must assure the match *between problem and an abstract solution derived from* solution classes. The match is found through structures' isomorphism and identical behaviour of corresponding resources. Further problem abstraction and structuring may be needed to assure it in explicit or implicit way. It is assumed, that some generic knowledge about problem solvability in the context of both solution classes is available. In the opposite case problem can be solved with an exhaustive search only. This is probably the strongest reason why a kind of systematic frame for the synthesis process is needed.

Synthesis procedures cannot be deterministic because of partial ordering of structure class. Structure limitation can play a dominant role in limiting the number of synthesis steps. The other limiting factor for the synthesis step count can be searched for in resource class. Resource class consisting of resources with simple behaviour characteristics will necessary complicate the synthesis. With proper problem structuring and composite resources built of resources from resource class this problem can be reduced. However, the effect of this depends on knowledge about problem characteristic properties.

Synthesis process can be stated as a combination of hierarchical problem decomposition and structuring. Hierarchical problem decomposition is a process basically opposite to abstraction. In our case it consists of refinement and interpretation processes. With its application problem is stepwise decomposed to a structure consisting of mutually dependent subproblems each of them have lower complexity with respect to the initial problem and subsequent subproblems obtained at earlier decomposition steps. This increases problem structure complexity and causes a consequence that structures of higher problem representation levels are replicated at lower representation levels. Applying structuring to arbitrary intermediate representation level results in structure change, while preserves representation level and behaviour.

Since hierarchical decomposition preserves structure and structuring preserves representation level it is clear that requirements expressed through solution classes can be met only if both hierarchical decomposition and structuring are applied during the synthesis process.

Synthesis process can be recognized as partial ordering relation between problem, intermediate solutions, and solution, where the solution or solutions are those intermediate solutions, which satisfy requests expressed through both solution classes.

To find a path between problem and solution and to avoid searching over the whole partially ordered structure of possible intermediate solutions, decisions based on their properties can drive the synthesis process. A minimal condition for an intermediate solution to be accepted for further synthesis is that it can be *expressed with the objects of both solution classes.* Different synthesis strategies can be used to determine the synthesis process. The simplest one and probably the less useful is one, where hierarchical decomposition to the resource class representation level is done first. This intermediate solution is then structured to assure compatibility with objects from structure class. However, structuring can become a task of excessive complexity within this approach. To avoid such situations, hierarchical decomposition and structuring are combined during the synthesis process. This assures manageable subproblems' complexity.

Applying structuring and abstraction to both solution classes match between intermediate solution and structure consisted of objects from both solution classes can be found at each intermediate representation level. When such a match can be found between an intermediate solution and a structure consisting of objects from both solution classes without any abstraction such intermediate solution is accepted as a solution. Structures' isomorphism and identical behaviour of corresponding resources assure systems' equivalence.

2. REPRESENTATIONS

Different views can be applied to represent the same system. In conventional one system is represented as a structure of interconnected resources with known behaviour. For system behaviour representation it is interesting to represent resource inputs with input positions and its outputs with output positions. Positions represent perception points from which object flow is oriented toward the resource or from it. For

two connected resources corresponding position plays double role. It is output position for one resource and input position for the other. Labelled bipartite directed graphs are suitable abstraction for this view on system's structure. System's behaviour can be represented with the behaviour of system resources. Based on resource positions collection of objects flowing to and from resource can be determined. Each collection corresponding to particular resource has objects represented with $m+n$ tuples corresponding to objects on input and output resource positions. Such a collection can naturally be represented with function or relation, depending whether resource reacts to the same input in deterministic or nondeterministic way. Relation between functional and relational resource behaviour will be clarified later in this section.

During the synthesis process more attention can be given to system structure since its behaviour remains the same throughout this process. For this reason resource structure can be represented neglecting positions in system structure.

Definition: Resource structure is labelled directed graph $G = (V,E)$, where $V = \{v_1,\ldots,v_n\}$ is a set of resources, and $E = \{e_1,\ldots,e_n\} \subseteq V \times V$ is a set of resource connections.

Resource labelling is defined with functions which assign labels to resources and connections and enable system behaviour recognition.

Returning to the initial system model represented with bipartite directed graph it can easy be recognized that this representation is in close connection with data-flow graphs defined in [1] , [2] . In our case modification of this definition will be used for representing system's structure and behaviour.

Definition: Controlled object-flow graph is labelled bipartite directed graph $G = (AULUK, E)$, $L \cap K = \emptyset$, $A = \{a_1,\ldots,a_n\}$ is a set of action nodes, and $LUK = \{l_1,\ldots,l_m, k_1,\ldots,k_r\}$ is a set of links, where $K = \{k_1,\ldots,k_r\}$ is a set of control links.
$E \subseteq (A \times (LUK) \cup (LUK) \times A)$ is a set of branches defined so, that the following restrictions are satisfied,

$$(a_i, l_k) \in E \,\&\, (a_j, l_k) \in E \Rightarrow a_i = a_j, \quad l_k \in LUK,$$

$$(l_k, a_i) \in E \,\&\, (l_k, a_j) \in E \Rightarrow a_i = a_j, \quad l_k \in LUK,$$

$1 \leq i, j \leq m,\ 1 \leq k \leq m + r.$

Two kinds of behaviour can be assigned to action nodes of controlled object-flow graph. One is represented with function, the other with controlled function. In the final case action node must have at least one input control link.

Let $f: X_1 \times \ldots \times X_n \longrightarrow Y$ be a function defined on known sets $X_1,\ldots,X_n$, Y. Controlled function g for function f is a function,

$$g: \text{Bool} \times X_1 \times \ldots \times X_n \longrightarrow Y, \quad \text{Bool} = \{0,1\}$$

$$g(p, x_1,\ldots, x_n) = \begin{cases} f(x_1,\ldots x_n) & ; \ p = 1 \\ \text{undef} & ; \ p = 0. \end{cases}$$

Figure 2.1 shows controlled object-flow graphs for functions f and g.

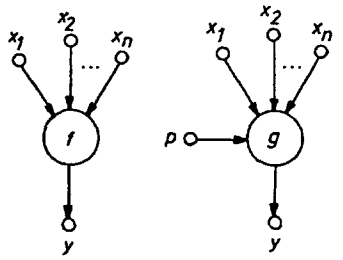


Figure 2.1: Controlled object-flow graphs for functions f and g

To avoid confusion instead of $g(0, x_1,\ldots, x_n) = \text{undef}$ we can define $g(0, x_1,\ldots,x_n) = z$, $z \in Y \cup \{z\}$. Object z has different interpretations in different implementation situations. In electronic systems it can be used to represent a high impedance condition, for example. In general, it represents no object or no activity situations.

The notion of controlled function can be extended to more sophisticated cases of controlled execution. Examples can be found in section 5.2. One further example is provided below,

$$g: A \times X_1 \times \ldots \times X_n \longrightarrow Y, \quad A = \{a_1, \ldots, a_m\}$$

$$g(a, x_1, \ldots, x_n) = \begin{cases} f_1(x_1, \ldots, x_n); a = a_1 \\ \ldots \\ f_m(x_1, \ldots, x_n); a = a_m. \end{cases}$$

However, it can be shown that they are expressible within the initial definition of controlled function. The notion of controlled function allows formal connection between controlled object-flow graph and control-flow graph.

Definition: Control flow graph is labelled bipartite directed graph $G = (QUA, E)$, where $Q = \{q_1, \ldots, q_n\}$ is a set of states, and $A = \{a_1, \ldots, a_m\}$ is a set of function nodes. $E \subseteq ((Q \times A) \cup (A \times Q))$ is a set of branches.

System's behaviour expressed through control flow graph is based on state concept. Particular state, when recognized active, fires corresponding functions. This situation is effectively modelled with a pair, consisting of decision function and controlled function, which is fired with decision function's value. Section 5 gives more detail about this topic.

To allow change of representation during the synthesis process conversions between the above representations are defined. To increase synthesis flexibility they are of one - to - many type.

Conversions between resource structure and controlled object - flow graph are based on graph isomorphism with deleting or inserting link nodes. However, those conversions are usually applied with

structuring, what makes correspondence between structures less explicit.

Conversions between controlled object - flow graph and control - flow graph are less evident. Figure 2.2 gives an example of controlled object - flow graph to control - flow graph conversion. Formal basis for those conversions is given in section 5.

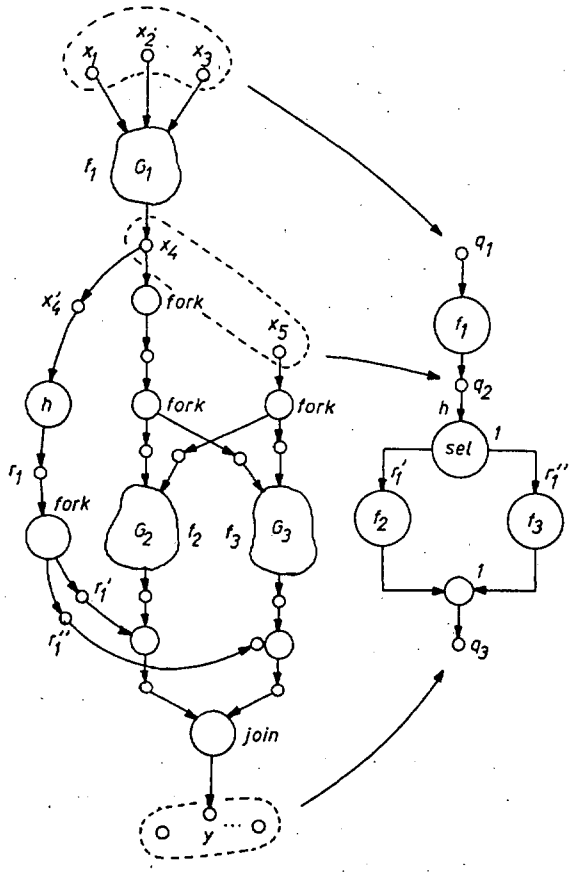


Figure 2.2: Example of controlled object - flow graph to control - flow graph conversion

Nodes of control - flow graph marked with $\perp$ are inserted to increase readability. From the formal point of view they are not necessary.

To describe system's behaviour production rules expressed with if then clauses can be used. We will show that such system descriptions can be converted to controlled object - flow graph descriptions and vice versa.

Originally conditional statements are used to represent if then clauses.

Take $R(x_1, \ldots, x_n) \longrightarrow Q(y_1, \ldots, y_m)$ as an example. Define characteristic functions for R and Q,

$$p = h_R(x_1, \ldots, x_n) = \begin{cases} 1 ; & R(x_1, \ldots, x_n) \\ 0 ; & \overline{R(x_1, \ldots, x_n)} \end{cases}$$

$$r = h_Q(y_1, \ldots, y_m) = \begin{cases} 1 ; & Q(y_1, \ldots, y_m) \\ 0 ; & \overline{Q(y_1, \ldots, y_m)}. \end{cases}$$

Implement h_Q with controlled function g_Q,

$$q = g_Q(p, y_1, \ldots, y_m) = \begin{cases} h_Q(y_1, \ldots, y_m); & p = 1 \\ 1 ; & p = 0. \end{cases}$$

It can easy be verified that p, r, and q determine truth table for conditional. Corresponding controlled object - flow graph is shown on figure 2.3.

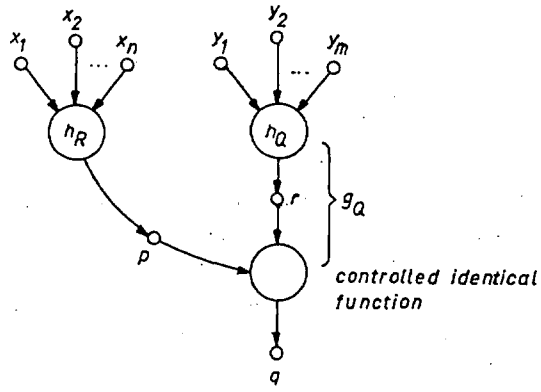


Figure 2.3: Controlled object - flow graph representing conditional clause

This representation allows inference processes' modelling in functional way.

Model of condition - action rule can be developed with slight modification of the above construction.

Let $R(x_1, \ldots, x_n) \longrightarrow Q(x_1, \ldots, x_n, y)$, and $Q(x_1, \ldots, x_n, y) \iff (y = f(x_1, \ldots, x_n))$.

Define characteristic function h for R, which defines domain of f.

$$p = h(x_1, \ldots, x_n) = \begin{cases} 1 ; & R(x_1, \ldots, x_n) \\ 0 ; & \overline{R(x_1, \ldots, x_n)}, \end{cases}$$

and implement f with controlled function g,

$$g(p, x_1, \ldots, x_n) = \begin{cases} f(x_1, \ldots, x_n) ; & p = 1 \\ \text{undef} ; & p = 0. \end{cases}$$

Controlled object - flow graph that corresponds to the above construction is shown on figure 2.4.

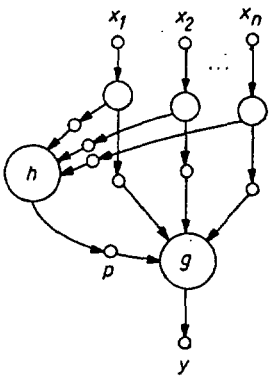


Figure 2.4: Controlled object - flow graph representing condition - action rule

It is interesting to note that observing behaviour only on input and output links of a graph we are not able to identify whether function f is implemented directly or with function g.

When dealing with systems it is usually presupposed that resources behave functionally. For systems described with relational connectives the above assumption may not be true. A minor modification allows that resources, which express relational behaviour can be treated in functional or relational way depending on point of view used. Since the complete treatment for arbitrary relations is relatively extensive, we will limit this presentation to binary relation R(x,y), which is partially closed with object a, R(a,y),

R(x,y)

⋮ ⋮
a b_i
a b_{i+1}
...
a b_{i+j}
...
a b_{i+n}
⋮ ⋮

Applying object a to a resource, which behaves corresponding to R, its response will be nondeterministic since it can delivers any object from b_i, b_{i+1}, ... ,b_{i+n} to its output position to satisfy R.

The behaviour of a such resource can be represented in functional way, if resource response is determined with a sequence function replacing R. Sequence function determines the order in which resource reacts with output objects to the same input determined with object a. Figure 2.5 gives tabular definition of sequence function and corresponding controlled object - flow graph for this case.

f:

x^k	y^{k-1}	y^k
a	b_i	b_{i+1}
a	b_{i+1}	b_{i+2}
...		...
a	b_{i+n}	b_i

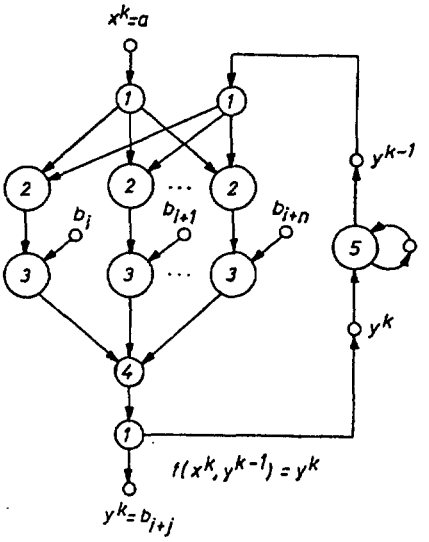


Figure 2.5: Tabular definition of sequence function and corresponding controlled object - flow graph

Identifiers 1 through 5 stand to identify functions fork, decision Boolean functions, controlled identical functions, function join, and memory function. Functions fork replicate input objects, decision Boolean functions control corresponding identical functions to deliver particular object b_{i+j} to function join, which is a treshold function and delivers object b_{i+j} to output link. Memory function assures necessary delay. Assumption is made that y^{k-1} has allways a value from $b_i,\ldots,b_{i+n}$. Two notes are necessary about the above presentation. First, it is simplified to serve conception presentation only, and second, sequence function can be defined in a number of different ways.

3. REFINEMENT AND INTERPRETATION

Using abstraction representation complexity can be reduced to a manageable level. During the synthesis process the situation is reversed, complexity grows, since this process is basically opposite to abstraction.

Assume a solution system represented with resource structure and resource behaviour. Using a substructure relation to determine an arbitrary substructure observe that its behaviour can be described with collection of objects which correspond to its input and

output positions. This enables that substructure is replaced with a resource having input and output positions that correspond to substructure input and output positions. The replacement causes lower system structure complexity.

Representing the system in each possible way with replacing substructures with resources while preserving their input output behaviour results in a class of systems with different structures and the same input output behaviour.

An important hypothesis can be made at this point. System structure abstraction has sense only if the substructure behaviour can be expressed with its input output behaviour[1]. Represent collections of objects which determine resources' behaviour with tables and assume they are of finite leught. Resource behaviour is represented with single table, while resource substructure behaviour is represented with a structure of interdependent tables. Select a pair resource, resource substructure with identical behaviour. The question arises whether the structure of interdependent tables can be developed based on the information obtained from single table.

Results of empirical study show that this is possible. However, for a formal proof of the above claim additional research is needed.

Results of mentioned study show that refinement based on the above concept is generally NP - complete problem since lower complexity bound grows exponentially with the number of table entries in non - trivial cases. Refinement is nondeterministic process that can be automated. Automatic algorithm development is possible. Controlled functions are necessary to obtain all possible refined versions of particular function. Refinement can be done on uncompletely specified tables. If this causes a lose of significant information, obtained algorithm will not be optimal.

Example 3.1 gives tabular refinement for selected case of binary addition and corresponding controlled object - flow graphs, which have no control links for this case. Only one path of the whole refinement process is presented.

Example 3.1

T

f:

a_1	b_1	a_o	b_o	c_2	d_1	d_o
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	0	1	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	1	0	0	1	0	0
1	1	0	1	1	0	1
1	0	1	0	0	1	1
1	0	1	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

G:

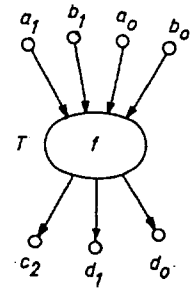
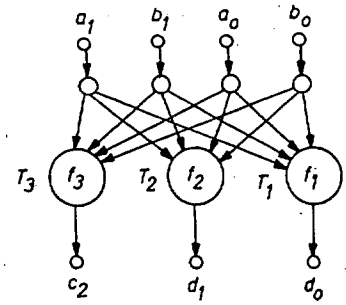


Table T represents binary addition decompose to tables T_1, T_2, and T_3, what results in the following controlled object - flow graph.



T_1

f_1:

a_1	b_1	a_o	b_o	d_o	
0	0	0	0	0	1.step
0	0	0	1	1	
0	1	0	0	0	2.step
0	1	0	1	1	
0	0	1	0	1	
0	0	1	1	0	
0	1	1	0	1	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	1	
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	1	
1	1	0	1	0	
1	1	1	0	1	
1	1	1	1	0	

T_{11}

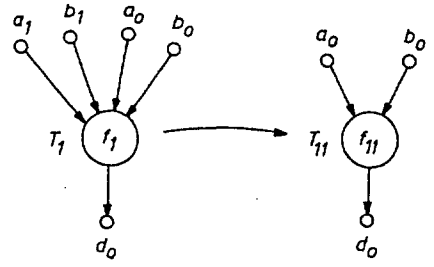
f_{11}:

a_o	b_o	d_o
0	0	0
0	1	1
1	0	1
1	1	0

T_1 becomes T_{11} after deleting redundant entries.

[1] Loops and circles in resource substructure can introduce additional substructure inputs and outputs which are local to it.

Corresponding controlled object - flow subgraph is reduced as represented below.



f_2:

a_0	b_0	a_1	b_1	d_1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

f_{21}:

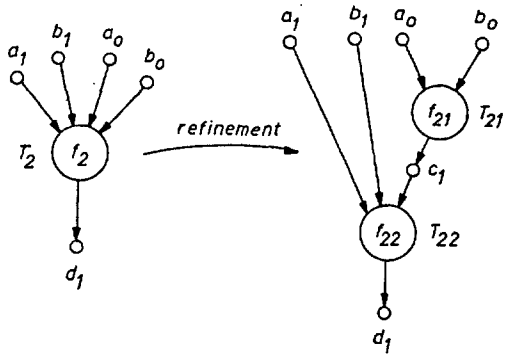
a_0	b_0	c_1
0	0	0
0	1	0
1	0	0
1	1	1

f_{22}:

a_1	b_1	c_1	d_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

T_2 is decomposed to T_{21} and T_{22}.

Corresponding controlled object - flow subgraph is refined as represented below.



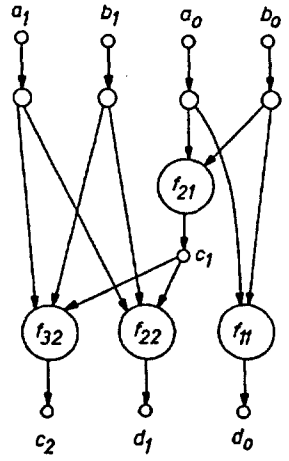
Refinement of T_3 is analogous to the refinement of T_2. The result is,

$T_{31} = T_{21}$,

f_{32}:

a_1	b_1	c_1	c_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Controlled object - flow graph obtained after refinement of T_1, T_2, and T_3 is given below.



Because of close relation between tabular and expressional function representation, the former can be thought of as tabular structure abstraction. Refinement procedures are generally developed for expressional function representation. Since this type of refinement is widely known its presentation will be avoided.

3.1 Function refinement

This subsection states conditions that have to be satisfied with function refinement. Figure 3.1 shows graph of relations and controlled object - flow graphs for functions f and g, which is composition of functions $g_1, \ldots, g_r$ obtained through refinement. Refined function g and initial function f have to satisfy the relation $f = h_1^{-1} \circ g \circ h_2 = g$, and h_1, h_2 are identical functions.

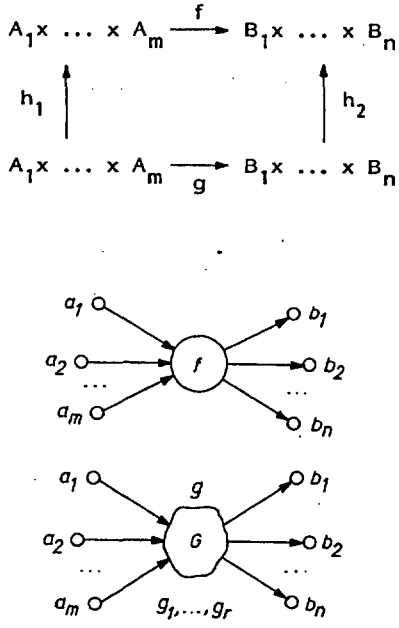


Figure 3.1: Function refinement

Dependent of actual representation defined in section 2 function refinement is completed with corresponding graph refinement. Procedures for them can be found in [2] , [3] .

Refinement can be thought of as a case of interpretation. However, we will give only basic definition and neglect this possibility.

Figure 3.2 shows graphs of relations and corresponding controlled object - flow graphs when interpreting function f with function g.

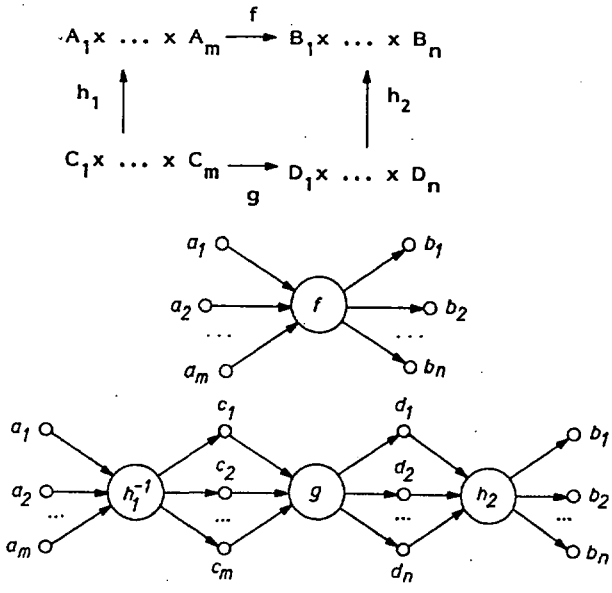


Figure 3.2: Interpretation of f with g

When interpreting function f with function g the relation $f = h_1^{-1} \circ g \circ h_2$, where h_1 and h_2 are surjective and possibly partial functions must be satisfied. Functions h_1 and h_2 assure compatibility with the environment. They can be stepwise removed with interpretation of environmental functions.

Since h_1^{-1} is generally relation this can cause formal inconsistency with controlled object - flow graph behaviour. Since h_1^{-1} can be represented with sequence function as illustrated in section 2 this can cause no serious problems. On the other hand h_1^{-1} and h_2 can be stepwise removed as mentioned above. The inconvenience can be avoided when h_1 is bijective.

3.2 Object interpretation and refinement

Until now objects were considered as integral units. However, for the reason of efficiency such observation is too restrictive. To avoid this, object representation can be adapted to problem representation level.

Intuitively, an arbitrary object composed of several objects can be viewed as integral unit or as a structure of objects positionally[2] determined. In first case the fact that object is composed is neglected. Several objects must satisfy some known relation which determine their mutual positions to be recognized as integral unit . To represent such structures n - tuples are used.

Object refinement and interpretation[3] are defined in connection with corresponding system's functions. Figure 3.3 shows a graph of relations and corresponding controlled object - flow graphs when refining objects of function f with objects of function g.

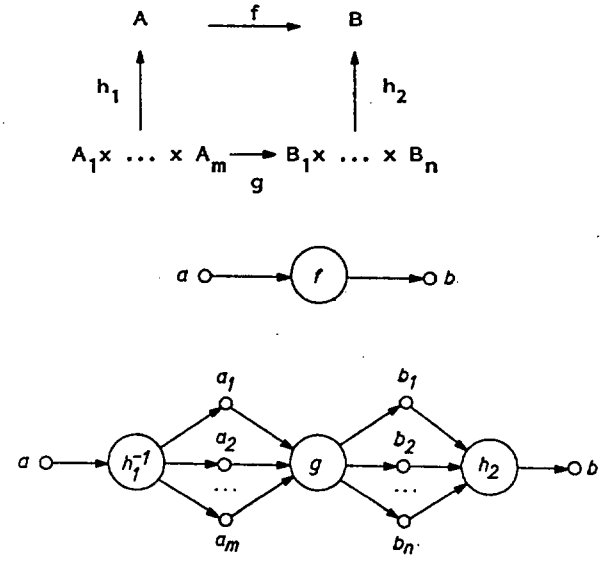


Figure 3.3: Object refinement and interpretation

[2] The notion of position is not restricted to physical position only.

[3] Difference between refinement and interpretation is semantical. Refinement is restricted to common semantic domain, while interpretation is not.

Object refinement and interpretation must satisfy, $f = h_1^{-1} \circ g \circ h_2$, where h_1 and h_2 are surjective and possibly partial functions. With further refinement functions h_1 and h_2 can be stepwise removed from the system.

Object refinement and interpretation can be extended to objects from $A_1, \ldots, A_m, B_1, \ldots, B_n$. Stepwise refinement of an object results in a tree structure, an example of which is shown on figure 3.4.

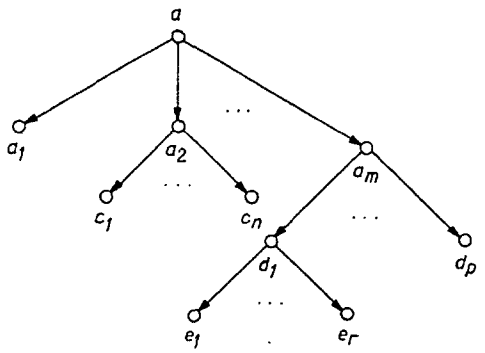


Figure 3.4: An example of object representation hierarchy

Refinement and interpretation have an interesting property, they preserve structure of previous higher represantation levels.

For illustration and application of this property assume an abstract system S_1 and a physical system S_2 and imagine both systems' behaviour in state spaces. Select an arbitrary state of S_1 and with interpretation and refinement determine corresponding state of S_2. This state will cause in S_2 a sequence of state changes. Assume a state from the sequence that has a corresponding state in S_1 and assume that this state is next state of initially selected state of S_1. The impression an observer seeing only the states of S_1 will have, is that S_1 changes states altough in reality state changes of S_1 are consequence of state changes in system S_2. Figure 3.5 illustrates the above explanation.

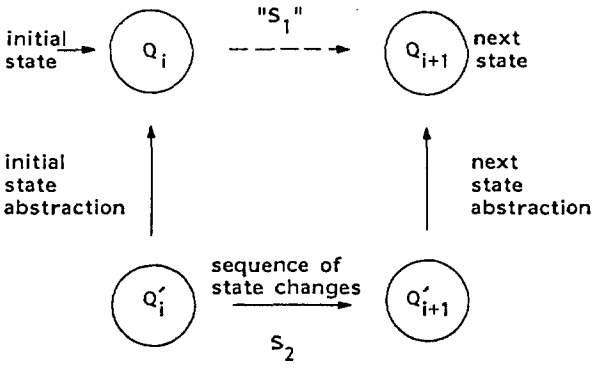


Figure 3.5: Implementation of S_1 with S_2

Refinement and interpretation were defined for all representations given in section 2. More details about them can be found in [2] and [3] .

4. STRUCTURING

Assume arbitrary system represented with resource structure. Select a resource and denote it with r. Since system representation level can vary such resource can represent resource substructure of lower representation level.

Determine those positions of system resource structure which carry objects from system input positions to resource r input positions. In the resource structure positions determine resource substructure which input positions are part of system resource structure input positions, while its output positions comprise resource r input positions.

For a subsystem, which belongs to the resource substructure determine composed relational expression denoted with E, which is satisfied when objects' state on resource r input positions enable resource activation. Instead with resource r function f determine its behaviour with relation corresponding to f and denote corresponding expression with Q. Form conditional $E \rightarrow Q$, which is satisfied after resource r delivers objects on its output positions. In the context of structuring it will be called control condition, since it allows uniform determination of each subsystem's state which causes resource r action.

Control condition can be rewritten as $E \rightarrow R$ & $R \rightarrow Q$, where R denotes relational expression describing states on resource r input positions, which cause its action. This form of control expression enables associative approach to control.

Determine characteristic function f_E for E,

$$f_E : A \longrightarrow \text{Bool}$$

$$f_E(a) = \begin{cases} 1 ; & E \\ 0 ; & \bar{E}, \end{cases}$$

where A stands for subsystem's states and Bool determines {0, 1} .

Characteristic function f_E will be interpreted as decision function since its value decides about resource r activation. Implement resource r function f with controlled function g as described in section 2. Join functions f_E and g into a pair. Collection of such pairs determined for each system's resource enables system behaviour and its structure reconstruction.

Figure 4.1 illustrates a distinction between the initial system and the system determined with collection of pairs decision function, controlled function.

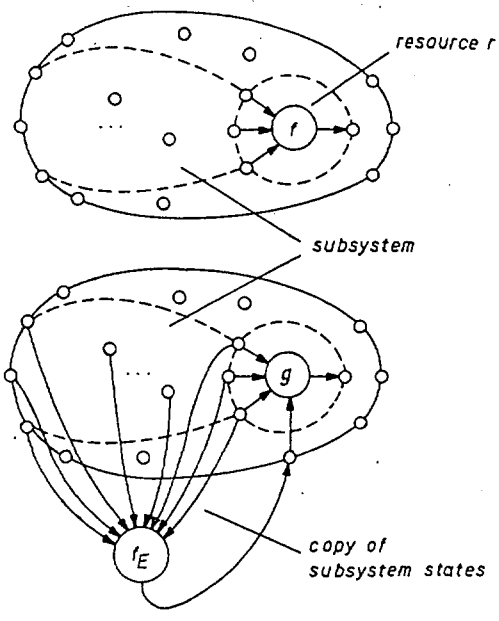


Figure 4.1: Distinction between systems

Define in A equivalence relation such that all states which activate the same subsystem's resource and resource r belong to the same equivalence class. Actually, more than one resource can correspond to particular equivalence class. Denote the set of equivalence classes with A/R and define bijective function from A/R to Q, where Q stands for the set of states. Based on each state q from Q it can be uniformly determined which resource(s) have to be activated as a consequence of this state, because bijective connection between A/R and Q.

Assume q from Q corresponds to equivalence class of states which causes resource r activation. Decision function corresponding to r can now be defined on set Q. To identify, when state q is active, what means that corresponding subsystem is in state which activates resource r, state transition function $f_s : Q \rightarrow Q$ is defined. It imitates subsystem state transitions from initial state to state corresponding to state q.

This model of control can now be expressed with the relations,

$$f_s(s)=q, \quad f_E(q)=1, \quad g(1, x_1, \ldots, x_m) = f(x_1, \ldots, x_m),$$

where s is the state preceeding state q, f_E is decision function, and g is controlled function corresponding to resource r function f.

Process of control as described above, can be defined for arbitrary system resources. There are no restrictions to limit it to a single state transition function. Extension of the model that state transition function enables n-way branching including loop control is relatively simple and will not be considered here. State approach to control is not the only one that can be developed based on decision function. In fact, as far as recognized all popular control strategies and combinations of them can be developed on that base.

One can easy recognize that if parts of control conditions are not necessary strictly distinct. This allows their logic composition, what results in decision function of the form,

$$f_E : A \rightarrow Bool^k, \quad k > 1.$$

More than simple resource can be controlled with a decision function and consequently more resources can be controlled with state transition function as mentioned earlier.

A pair decision function, controlled function enables structuring. A set of such pairs which determines the system can be partitioned to disjunctive subsets. The same effect can result from system partition on the set of disjunctive subsystems.

System integration based on its arbitrary partition can be realized in more different ways. Before do this we have to develop object transfer functions and memory functions. Transfer functions realize object transfer between positions, while memory functions assure that objects retain particular positions so long as determined with control structure.

4.1 Transfer functions

Flow of objects between two arbitrary subsystems separated with a partition can be implemented with selecting and distributive functions.

Assume a subsystem which has to be connected to another subsystem over positions $p_1, \ldots, p_n$. Denote sets of objects corresponding to positions with $X_1, \ldots, X_n$, and let $X = X_1 \cup \ldots \cup X_n$.

Selecting function Π is defined as,

$$\Pi : N \times X_1 \times \ldots \times X_n \rightarrow X$$

$$\Pi(i, x_1, \ldots, x_n) = \begin{cases} x_i, & 1 \leqslant i \leqslant n \\ \text{undef}, & \text{otherwise.} \end{cases}$$

The set of natural numbers N above can be replaced with arbitrary linearly ordered set. Indexes of $x_1, \ldots, x_n$ become objects of such set. Linear ordering can be avoided if selecting function is defined in tabular form. Both approaches to define selecting operation can be mixed. Levels of indirection can be built to the above definition to determine particular position of n-tuple.

Figure 4.2 shows controlled object - flow graph of selecting function.

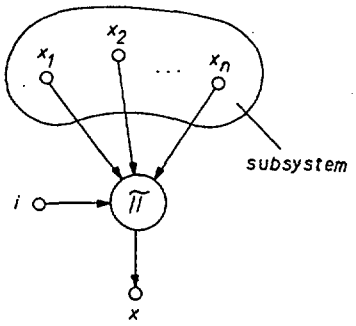


Figure 4.2: Controlled object - flow graph of selecting operation

Distributive function delivers objects obtained from selecting function to prespecified positions of a subsystem.

Distributive function is defined as,

$$\delta : N \times X \longrightarrow Y_1 \times \ldots \times Y_m$$

$$(y_1, \ldots, y_m) = \delta(j,x) = \begin{cases} (\text{undef}, \ldots, x, \ldots, \text{undef})^4, y_j = x, \\ \qquad\qquad 1 \leq j \leq m \\ \text{undef}, \quad \text{otherwise.} \end{cases}$$

Variables $y_1, \ldots, y_m$ correspond to subsystem input positions. Pair (i,j) determines a path from output position of one subsystem to input position of another subsystem. Composition of Π and δ can then realize arbitrary connenction between subsystems.

Figure 4.3 shows controlled object - flow graph of distributive function.

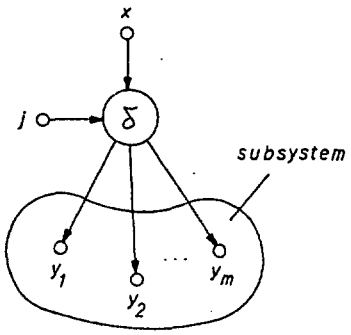


Figure 4.3: Controlled object - flow graph of distributive function

Selecting and distributive functions can be composed to realize arbitrary complex networks, which are capable of transfering specified amount of objects in time and space.

More requests for transfer than transfer paths available can exist simultaneously. Such requests are paid with appropriate time - space partition of transfer.

Selecting and distributive functions are not the only transfer functions possible. However, they are sufficient to implement arbitrary connection between two subsystems. They can be decomposed and expressed with controlled function defined in section 2.

4.2 Object representation in recursive domain

Let X be a set of objects to be represented in recursive domain. This representation is achived with function,

$$f : N \times X \longrightarrow X$$

$$f(i, x) = y$$

Pair (i, x) is represented with the notation x^i.

$$f(x^i) = \begin{cases} x, & n_1 \leq i \leq n_2, \quad n_1, \ n_2 \text{ from N} \\ \text{undef}, & \text{otherwise.} \end{cases}$$

The value of x is represented in domain which is linearly ordered set. Because of simple notation the set of natural numbers was selected, otherwise a set of states can be used, since it is linearly ordered with the state transition function.

In the real systems objects may have the ability to retain particular position - objects in mechanical systems for example have such property. However, since this is generally not true memory property have to be assured for objects not having this property.

It is defined in recursive domain with the function,

$$f : N \times X \longrightarrow N \times X$$

$$y^{i+1} = f(x^i) = x^i, \quad n_1 \leq i < n_2.$$

In the language of state transitions it has the ability to assure object position from current state to next state.

This limitation can be removed with memory function control. Let p^i be the value from Bool of decision function at i from N. Then,

$$y^{i+1} = g(p^i, x^i, y^i) = \begin{cases} x^i, \ p^i = 1, & n_1 \leq i < n_2 \\ y^i, \ p^i = 0, & n_1 \leq i < n_2. \end{cases}$$

For $k \geq 1$ and $p^i = 1$ for particular i only, y^{i+k} will have the value of x^i, if $i+k < n_2$.

Extension to represent an object in time domain is straightforward and will not be considered here.

4.3 Function representation in recursive domain

System's behaviour can be represented in recursive domain when its functions are transformed to this domain. Similar extension in representation can be developed for time domain, continous or discrete. This allows synthesis in time domain.

4 Recall a no object situation described in section 2.

Let $f : X_1 x \dots x X_m \longrightarrow Y$ be a function to be represented in recursive domain.

Define object interpretation functions,

$h_1 : N \times X_1 \times \dots \times X_m \longrightarrow X_1 \times \dots \times X_m$

$h_1 (i, x_1, \dots, x_n) = (x'_1, \dots, x'_m)$.

Represent $(i, x_1, \dots, x_m)$ with $(x^i_1, \dots, x^i_m)$, and define

$$h_1 (x^i_1, \dots, x^i_m) = \begin{cases} (x_1, \dots, x_m); & n_1 \leq i < n_2 \\ \text{undef}; & \text{otherwise.} \end{cases}$$

$h_2 : N \times Y \longrightarrow Y$

$h_2 (i, y) = y'$

Represent pair (i,y) with y^i and define,

$$h_2 (y^i) = \begin{cases} y; & n_1 < i \leq n_2 \\ \text{undef}; & \text{otherwise.} \end{cases}$$

Function f can now be represented in recursive domain with function g,

$g : N \times X_1 x \dots x X_m \longrightarrow N \times Y$

$g (i, x_1, \dots, x_m) = (i+1, y)$ or,

$g (x^i_1, \dots, x^i_m) = y^{i+1}$

$$g (x^i_1, \dots, x^i_m) = \begin{cases} f(x_1, \dots, x_m); & n_1 \leq i < n_2 \\ \text{undef}; & \text{otherwise.} \end{cases}$$

Function value is determined with i+1, while domain values are determined with i. This assures memory property. In fact, function g is a composition based on function f and memory function. If defined as controlled function its value can be retained arbitrary long. Figure 4.4 shows controlled object - flow graph for such case.

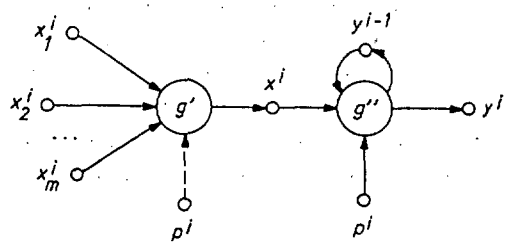


Figure 4.4: Controlled object - flow graph for controlled implementation of g

With the above developments we have minimal tools to define basic structuring concepts.

4.4 Distributive structuring

Assume a system determined with the pairs decision function, controlled function at an arbitrary level of representation. Define a partition of pair set. Each subset of the pair set determines a subsystem, unconnected in general. Define at least one input and at least one output position for each subsystem. Select two arbitrary subsystems and determine all connections between them with regard to unpartitioned system. Between subsystems define transfer functions to reconstruct connections determined with unpartitioned system. At this point representation should be changed to recursive domain in general, since resource sharing is introduced. A sketch of the above development is shown on figure 4.5.

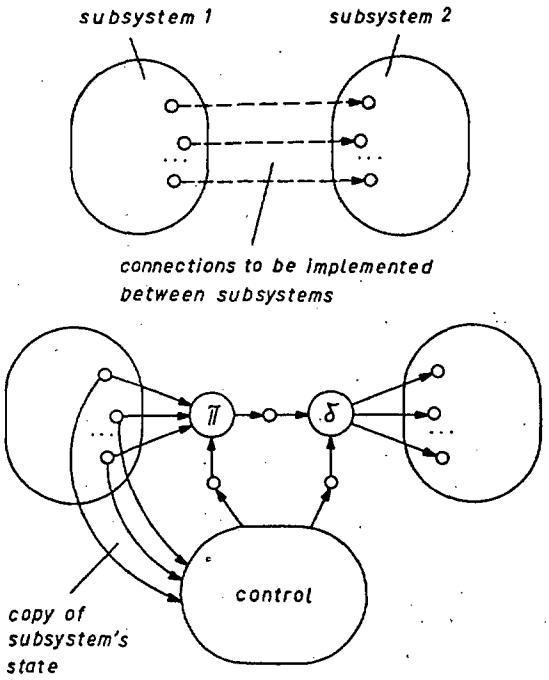
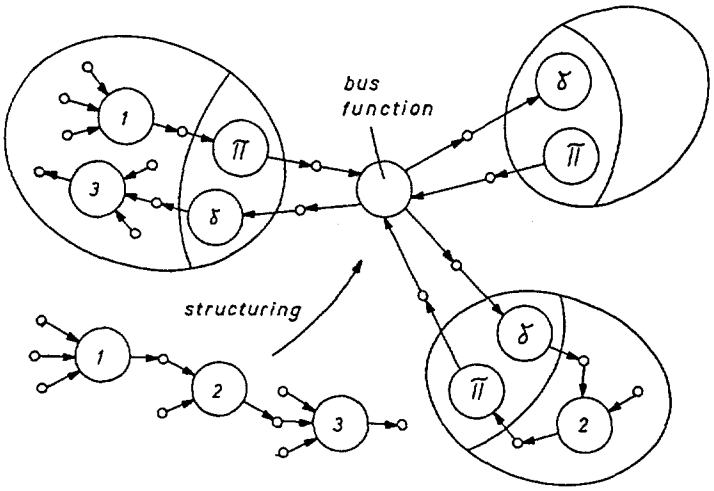


Figure 4.5: A sketch for subsystem connection

The process of developing connections is repeated for all subsystems. Analogous approach is used to develop connections with the system environment. With the insertion of transfer functions which are compositions of selecting, distributive and memory fuctions additional system states are introduced. Decision function which controls particular resource should take this into account to retain compatibility.

Distributive structuring results in a system organized around more or less tightly connected subsystems and preserves the behaviour of the initial system. Figure 4.6 shows a simplified example of a system distributively structured.



Note: Control structure is not shown.

Figure 4.6: Example of distributive structuring

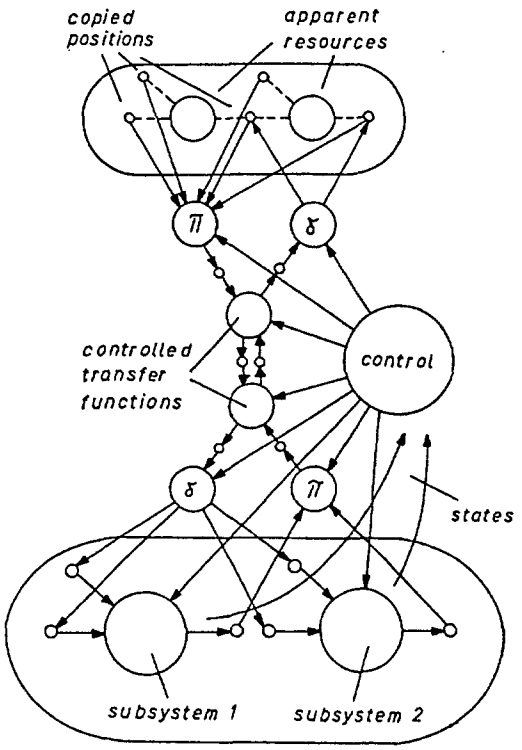


Figure 4.7: Example of hierarchical structuring

4.5 Hierarchical structuring

Assume a system represented with a set of pairs decision function, controlled function. Define a partition, such that each subsystem determined with it is connected. Determine a copy of all those positions, which with the system partition decay to input and output positions. Those positions are actually subsystems' input and output positions. Each copied position will be during structuring process connected to corresponding subsystems' input and output positions.

For each pair consisting of input and output positions determined with the partition and corresponding copied position define transfer function, which enables object transfer from output position over copied position to input position. To connect two subsystems over corresponding copied positions a composition of selecting, memory, and distributive functions is generally needed.

Since objects on copied positions represent system's state at higher level of representation than the initial representation level the behaviour of hierarchically structured system can be interpreted in the context figure 3.5. Transfer functions can namely be completed with object abstraction and interpretation functions. This assures compatibility in representation levels.

The approach to hierarchical structuring can be upgraded to state hierarchy, which is similar to memory hierarchy in contemporary systems.

Similar as with distributive structuring transfer functions introduce additional system's states. Because of this decision functions, which control the execution must be appropriately modified.

Figure 4.7 shows a simplified example of hierarchically structured system.

Both structuring techniques can also be applied on system's control, transfer, and memory structures. Distributed and hierarchical structuring can be combined and applied at arbitrary system representation level. As far as recognized, the proposed structuring techniques are sufficient to model arbitrary system architecture. They were developed for all representations given in section 2. To structure a system in particular representation this can be done without representation change. Since structuring allows resource sharing, system's synthesis can respect cost and performance requirements. Based on the given approach to structuring system's synthesis and software development cannot be separated since they are tightly connected with structuring. This gives at least theoretical possibility for automatic software development. On the other hand highly structured systems can be developed without any control code or software in the usual meaning.

5. CONCLUSION

Review of system's synthesis process is given. Since this topic is very extensive this presentation is focused on these domains estimated as significant. However, domains determined with real - time, fault tolerance, and intelligent behaviour paradigms were completely avoided in the presentation. This does not mean that systems from these domains cannot be developed within the proposed context. In contrary,

some significant practical results were obtained in the synthesis of hard real - time systems and fault tolerance in the domain of industrial process control systems. At the same time it was shown that structuring is still very controverse notion with diverse span of significance, although it is more or less clear that hard real - time and fault tolerance paradigms are of little use in lousy structured domains. Similarly, fault tolerance cannot be a compensation for poor system design. Those were some of the reasons why structuring was given such attention in the presentation.

Since it becomes more clear, that differences caused with separate development of software systems, software engineering, artificial intelligence, knowledge engineering, etc., are caused mainly because of diverse views to problems and that their solution can only be achieved with multidisciplinary approach, latest efforts to avoid such situation result in systems engineering approach.

Based on this approach, systems which are capable to learn particular behaviour, analyse it and construct systems that behave equivalent can be synthesized, based on the proposed approach to the synthesis process.

6. REFERENCES

1 K.M. Kavi, B.P. Buckles, U.N. Bhat
A Formal Definition of Data - Flow Graph Models, IEEE Trans. on Computers, p. 940-948, No. 11, Vol. C-35, Nov. 1986.

2 K.M. Kavi, B.P. Buckles, U.N. Bhat
Isomorphism Between Petri Nets and Dataflow Graphs, IEEE Trans. on Software Eng., p. 1127-1134, No. 10, Vol. SE-13, Oct. 1987.

3 M. Gerkeš
Structures and Models, Resource Interconnection, Functional Behaviour, and Control, Report, Metalna, 1988.

4 M. Gerkeš
Funkcionalno modeliranje sistemov, Strukturiranje, Poročilo, Metalna 1988.