Vol. 0, No. 0 (1998) 000–000
©

MULTIPLE VIEWS ANALYSIS OF SOFTWARE DESIGNS

BOUMEDIENE BELKHOUCHE*
EECS Department, Tulane University
New Orleans, LA 70118, USA

and

CUAUHTÉMOC LEMUS–OLALDE
EECS Department, Tulane University
New Orleans, LA 70118, USA

Received (received date)
Revised (revised date)
Accepted (October 1998)

Designs are expressed in terms of structure, behavior, objects, modules, and functions which, once consolidated, become design blueprints. This paper describes a formal framework for expressing, analyzing, and comparing multiple views of designs. The analysis of multiple designs or views using different design notations is proposed as a strategy to enhance design quality by providing a systematic identification of design defects and discrepancies. Views are formalized, analyzed independently, and then compared to each other. The type of design discrepancies identified are omission of information, incompatible information, and inconsistencies between views.

Keywords: Design analysis, formal design representation, multiple views, abstract interpretation.

1. Introduction

In the process of modeling a system, the software designer generates a set of designs each of which captures what may be deemed pertinent features of the system (Figure 1). This simple approach to generating several perspective-based blueprints or *multiple views* to provide a multi–angled understanding of a problem has been employed routinely and successfully in other engineering fields. For example, architects and civil engineers develop various plans each representing a view (e.g., dead-load distribution, electrical wiring layout, air conditioning layout, etc.) that emphasizes some particular aspects. Thus, a view can be thought of as a model describing a subset of the relevant attributes of the system.

The focus of this research is to establish a formal framework for expressing, comparing, and analyzing multiple views of designs in order to derive complementary and contradictory information. The views are expressed in two different *design notations* [1]. Specifically, we consider a functional view captured by a Data Flow Diagram (DFD) notation, and a functional/structural view captured by a Structure

*Tulane University EECS Department, New Orleans, LA 70118

Chart (SC) notation. An essential part of our work was dedicated to the definition of a formalism in which the views and notations were formalized and analyzed. Second, the development of a systematic and formal approach to provide multiple views analysis support at the design phase was elaborated.

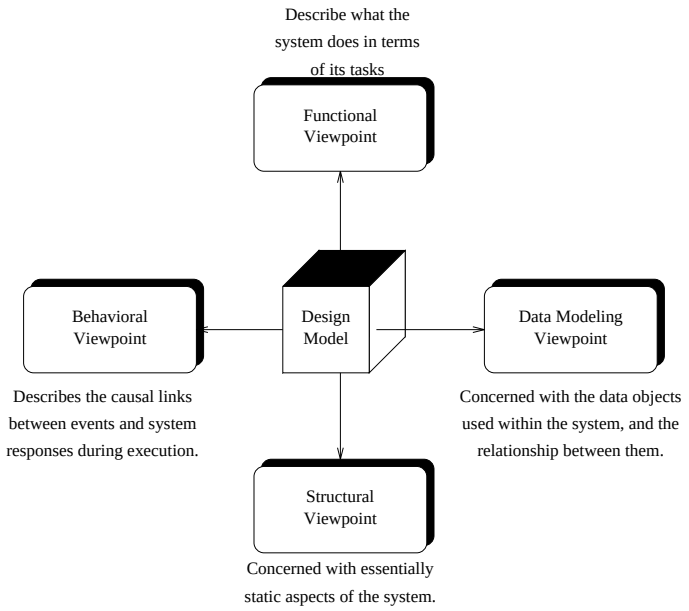

Figure 1: A design model may be captured by different viewpoints.

While multiple views analysis is a subject of active research, our approach is novel and unique in integrating the following two aspects: (1) its emphasis on design methods; and (2) its use of well-established formal foundations.

The paper is organized as follows. Section 2 is an overview of the state of the art regarding the use, manipulation, integration and analysis of multiple views. In section 2, , a formal representation of graphical designs is described. In section 4, our approach to multiple views analysis is introduced and explained. In section 5, the analysis process is described. In section 6, an example to illustrate our approach is covered. Section 7 concludes the paper.

2. Related Work

The use of Multiple Views in software development is not a new technique. For instance, datagrams and actigrams in SADT define a multiple views design [2]. Recently, research in multiple views has focused on making the approach explicit by addressing issues related to notations, analysis, tools integration, and benefits. Multiple views have been used in requirements elicitation [3], domain modeling,

requirements definition and specification [4], program development systems [5], and software design reuse [6].

The *viewpoint resolution technique* is used to validate requirements [3]. A language is used to express different viewpoints, and a set of analogy heuristics is devised to analyze them syntactically. The heuristics help detect missing information and conflicting information. Also, a classification (hierarchies) among several viewpoints is provided. The analysis process, and the manipulation and representation of knowledge are supported with automated tools [7]. *Domain modeling with hierarchies of alternative viewpoints* bases its modeling strategy on developming a hierarchy of conflicting viewpoints representations [7]. As the elicitation process proceeds, hierarchies among the views are generated. Lower levels of a hierarchy contain conflicting and uncertain viewpoints. Viewpoint creation and manipulation is provided by an automated analyzer. *The viewpoint framework* addresses the problem of multiple perspectives, which includes factors such as the development of composite systems, an environment with many participants, several representation schemes, diverse domain knowledge, and differing development strategies [4]. The viewpoint framework provides a basis for dealing with the structure, organization and management of different perspectives, consistency among perspectives, and concurrent collaboration among participants. The representation and integration of multiple views using a canonical graph representation (*semantic program graphs*) is proposed in [8]. This is in essence a new intermediate design notation that acts as a common notation to express other notations.

3. Formal Representations of Graphical Designs

A simple formal representation of design notations is required to facilitate the analysis of views. General formal representations of graphical designs can be found in [9, 10, 11]. Formal representations specific to data flow diagrams are given in [12, 13]. To illustrate our multiple views analysis framework, we concentrate our discussion on two design methods. The Data Flow Diagram (DFD) notation and Structure Chart (SC) notation were selected based on the following simple criteria:

Design Constructs. Number of constructs used in the notation to represent a design (i.e. five or six constructs).

Level of Abstraction. The role of each notation is identified in terms of level of abstraction for supporting modeling during design stages (i.e. architectural, detailed).

Basic design constructs that a notation employs in the construction of a model or view.

Viewpoint that a notation emphasizes during attribute abstraction.

3.1. *DFD Graph Representation*

- **Basic Design Constructs**
  - the circle (bubble), which is used to denote an operation, and is labeled with the name of the operation. It depicts a process;
  - the box, used to denote an external source or sink of information;
  - the parallel bars, used to denote a data store or file;
  - the arc, used to denote the flow of information between the other three components.
- **Viewpoint**: Functional.
- **Design attributes captured**: Information flow, dependency of operations on other operations, relation with data stores.

A DFD graph is a quintuple:

$$DFD = (Nodes, \sigma, \pi, \lambda, Arcs),$$

where:

- *Nodes* is a finite set of nodes. Each node represents:
  - Function (activity or process).
  - Data store.
  - External Entity (Sink or Source).
- *Arcs* is the set of edges representing data flows. It is a binary relation on *Nodes*: $Arcs \subseteq Nodes \times Nodes$.
- The refinement function σ is as defined as: $\sigma : Nodes \rightarrow 2^{Nodes}$. It assigns to each node $n \in Nodes$ its set $\sigma(n)$ of refinements and is restricted to being cycle free. We speciliaze it as follows:
  - $\sigma_{function} : Nodes \rightarrow 2^{Nodes}$,
  - $\sigma_{store} : \emptyset$, $\sigma_{entity} : \emptyset$, $\sigma_{connector} : \emptyset$; that is, store, entity, and connector are not subject to refinement in this context.
- The partitioning function π is defined as: $\pi : Nodes \rightarrow 2^{Nodes \times Nodes}$. It associates with each node $n \in Nodes$ some equivalence relation $\pi(n)$ on the set of refinements, $\sigma(n)$. This is just a rigorous way of specifying the decomposition of n into its orthogonal components, which are now defined simply to be equivalence classes indicated by the relation $\pi(n)$ denoted by $\pi_1(n), ..., \pi_{k_n}(n)$.
- The data flow refinement function λ_{di} is defined as $\lambda_{di} : Arcs \longrightarrow 2^{Arcs}$. It assigns to each edge $d \in Arcs$ its set $\lambda(n)$ of data flow refinements according to the information contained in the data dictionary.
- Each node has an identification label, *l–node:* $Nodes \rightarrow Ln$, where *Ln* is a set of node labels.

- Each arc has an identification label, *l–arc:* $Arcs \rightarrow 2^{La}$, where La is a set of arc labels.
- Each edge is assigned a unique identification such as *d1*, *d2*, etc. *id–arc:* $Arcs \rightarrow 2^{Id}$, where Id is a set of identification numbers.
- Each node has a specific role, *r–node:* $Nodes \rightarrow Rn$, where Rn is a a set of node roles.
- Each arc has a particular role, *r–arc:* $Arc \rightarrow Ra$, where Ra is a set of arc roles.

3.2. *SC Graph Representation*

- **Basic Design Constructs**
  - the box, which denotes a procedure (subprogram);
  - the arc, which denotes invocation;
  - the side arrows, which provide information about data flow in terms of parameter passing.
- **Viewpoint**: Functional and Structural.
- **Design attributes captured**: Invocation hierarchy between procedures, decomposition into procedures.

A SC graph is a quintuple:

$$SC = (Nodes, \sigma, \pi, \lambda, Arcs),$$

where:

- *Nodes* is a finite set of nodes. Each node represents:
  - Module descriptor (procedure or subprogram)
- *Arcs* is the set of edges.
- The functions σ, π, λ and the syntactical definitions *l–node*, *l–arc*, *id–arc*, *r–node*, *r–arc* are as defined earlier.

4. Multiple Views Analysis

In our framework, multiple view analysis (MVA) is the second stage of the analysis process. In the first stage, an independent analysis of each view is automatically performed [14]. This process consists of a global analysis (detection of cycles in the decomposition process of nodes), a structural analysis (use of precedence relations to check for the balancing rule), and a completeness analysis (verification of design

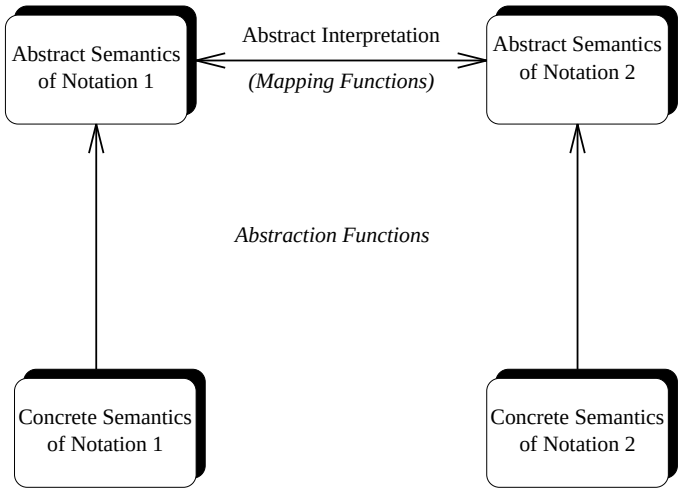

Figure 2: Multiple View Analysis Framework.

validity). These analyses are performed to guarantee that each design is well–constructed [15, 13]. Well–constructed designs are subsequently subjected to the multiple view analysis.

Given the wide range among design views, it is practically infeasible to perform MVA at the concrete representation level. Thus, some form of abstraction must be carried out before engaging in MVA. Our approach to abstraction is derived from the abstract interpretation method [16] (see Figure 2). The starting point of the framework is a concrete representation which associates with each Data Flow Diagram (D_{DFD}) and Structure Chart (D_{SC}) a formal graph representation of the corresponding designs. To facilitate the abstract representation analysis, an abstraction function is defined based on common features captured by both types of designs. These common features are the input and output relationships of the designs. The construction of the abstract models is then derived by defining abstraction and mapping functions. Consider figure 3, where $D1$ corresponds to a concrete design one, and $D2$ corresponds to a concrete design two (D1 denotes D_{DFD} and D2 denotes D_{SC}). The abstraction of the concrete design is obtained by applying an abstraction function α (α_1 for $D1$ and α_2 for $D2$). In both cases, the abstract design ($AD1 = \alpha_1(D1)$, $AD2 = \alpha_2(D2)$) is constructed from the input and output data flow sets abstracted by the input/output functions, I(n) and O(n) respectively. The function α isolates certain attributes of the concrete design, and thus "forgets" other attributes. Mapping functions f and g transform an abstract design from one representation domain to another. A few definitions are required to proceed.

Definition 1. Most Relevant nodes and arcs. The most relevant nodes of a design D are those whose roles are classified as a *process* in a Data Flow Design

(D_{DFD}) or *module* in a Structure Chart Design (D_{SC}). The most relevant input arcs in a D_{DFD} are those whose roles are *data flows*, and whose roles in a D_{SC} are *data couples*, and they originate at a relevant node. The most relevant output arcs in a D_{DFD} are those whose roles are *data flows*, and whose roles in a D_{SC} are *data couples*, and they terminate at a relevant node.

Definition 2. A design *D1* has less information (weaker) than a design *D2*, denoted as $D1 \sqsubseteq D2$ if and only if

- *(i)* $|\text{R–Nodes}_{D1}| \leq |\text{R–Nodes}_{D2}|$
  where $|\text{R–Nodes}_{Di}|$: cardinality of the set of relevant Nodes of design Di.
- *(ii)* $|\text{R–Arcs}_{D1}| \leq |\text{R–Arcs}_{D2}|$
  where $|\text{R–Arcs}_{Di}|$: cardinality of the set of relevant Arcs of design Di.
- *(iii)* $\text{R–l–node}_{D1} \subseteq \text{R–l–node}_{D2}$ and $\text{R–l–arc}_{D1} \subseteq \text{R–l–arc}_{D2}$
  where $\text{R–l–node(arc)}_{Di}$: labels set of the relevant nodes (arcs) of design Di.
- *(iv)* $\text{R–r–node}_{D1} \subseteq \text{R–r–node}_{D2}$ and $\text{R–r–arc}_{D1} \subseteq \text{R–r–arc}_{D2}$
  where $\text{R–r–node(arc)}_{Di}$: roles set of the relevant nodes (arcs) of design Di.

Definition 3. Two designs D1 and D2 are said to be graphically-abstract (*ga*) with respect to each other if there is a one-to-one correspondence between the input/output data flows such that the input/output relations for their most relevant nodes are preserved.

Definition 4. The set of input data flows of a node $n \in Nodes.$, $I(n) = \{\langle x, n\rangle \mid \langle x, n\rangle \in Arcs\}$ is the set of data flows that go into n. Similarly, the set of output data flows of n, is defined by $O(n) = \{\langle n, y\rangle \mid \langle n, y\rangle \in Arcs\}$.

Definition 5. The input relation of a design D is the set of input data flows of all nodes that are contained in D.

$$input_D = \{\langle x, y\rangle \,|\, x \in Nodes_D, y \in I_D(n)\}$$

Definition 6. The output relation of a design D is the set of output data flows of all nodes that are contained in D.

$$output_D = \{\langle x, y\rangle \,|\, x \in Nodes_D, y \in O_D(n)\}$$

4.1. Mappings

Figure 3 depicts the mappings needed in the Multiple View Analysis.

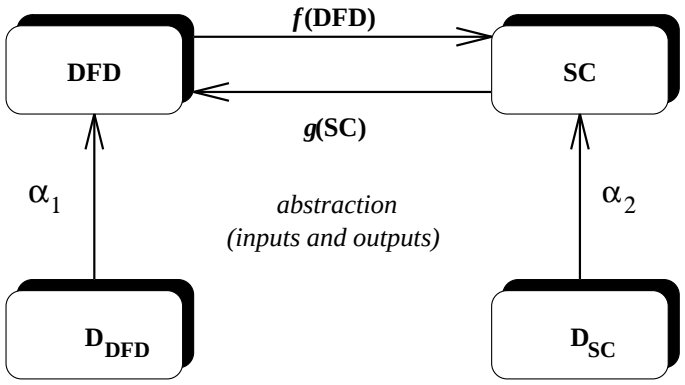

Figure 3: Abstraction and Analysis Functions.

Definition 7. Transformation: $f: \ D_{DFD} \longrightarrow D_{SC}$.

Let D be a Data Flow Design, denoted D_{DFD}. Let I_{DFD} and O_{DFD} be the input and output relations of D_{DFD}.

$$f_{input} : input_{DFD} \longrightarrow input_{SC}$$

$$I_{SC}(n) = (\bigcup f^1_{input}) \ \cup \ (\bigcup f^2_{input})$$

$$f^1_{input} = \begin{cases} d & \text{if } d \in I_{DFD}(n), \\ & (< d_x, d_y >, d) \in \text{id–arc} \wedge \\ & (d_x, role) \in \text{r–node} \wedge \\ & role = Process \ \vee \ role = Connector \\ & \text{(originates at Process or Connector)} \\ \emptyset & \text{otherwise} \\ & \text{(originates at Entity or Store)} \end{cases}$$

$$f^2_{input} = \begin{cases} d & \text{if } \sigma_n \neq \emptyset, \forall i \in \sigma'_n, d \in O_{DFD}(i), \\ & (< d_x, d_y >, d) \in \text{id–arc} \wedge \\ & (d_y, role) \in \text{r–node} \wedge \\ & role = Process \vee role = Connector \\ & \text{(terminates at Process or Connector)} \\ \emptyset & \text{if } \sigma_n = \emptyset \\ & \text{(terminates at Entity or Store)} \end{cases}$$

$$f_{output} : output_{DFD} \longrightarrow output_{SC}$$

$$O_{SC}(n) = (\bigcup f^1_{output}) \ \cup \ (\bigcup f^2_{output})$$

$$f^1_{output} = \begin{cases} d & \text{if } d \in O_{DFD}(n), \\ & (< d_x, d_y >, d) \in \text{id–arc} \wedge \\ & (d_y, role) \in \text{r–node} \wedge \\ & role = Process \ \vee \ role = Connector \\ & \text{(terminates at Process or Connector)} \\ \emptyset & \text{otherwise} \\ & \text{terminates at Entity or Store)} \end{cases}$$

$$f^2_{output} = \begin{cases} d & \text{if } \sigma_n \neq \emptyset, \forall i \in \sigma'_n, d \in I_{DFD}(i), \\ & (\langle d_x, d_y \rangle, d) \in \text{id–arc} \wedge \\ & (d_x, role) \in \text{r–node} \wedge \\ & role = Process \vee role = Connector \\ & \text{(originates at Process or Connector)} \\ \emptyset & \text{if } \sigma_n = \emptyset \\ & \text{(originates at Entity or Store)} \end{cases}$$

where,

$\sigma'_n = \{n | n \in \sigma_n \ \wedge \ \langle n, role \rangle \in \text{r–node} \ \wedge \ role = Process\}$

$input_{SC}$: denotes $I_{SC}(n), \ \forall n \in D_{SC}$

$output_{SC}$: denotes $O_{SC}(n), \ \forall n \in D_{SC}$

Definition 8. Transformation: $g: \ D_{SC} \longrightarrow D_{DFD}$.

Let D be a Structure Chart design, denoted D_{SC}. Let I_{SC} and O_{SC} be the input and output relations of D_{SC}.

$$g_{input} : input_{SC} \longrightarrow input_{DFD}$$

$$I_{DFD}(n) = g_{input}$$

$$g_{input} = \begin{cases} I_{SC}(n) & \text{if } \sigma_n = \emptyset \\ I_{SC}(n) - \bigcup_{\forall i \in \sigma_n} O_{SC}(i) & \text{otherwise} \end{cases}$$

$$g_{output} : output_{SC} \longrightarrow output_{DFD}$$

$$O_{DFD}(n) = g_{output}$$

$$g_{output} = \begin{cases} O_{SC}(n) & \text{if } \sigma_n = \emptyset \\ O_{SC}(n) - \bigcup_{\forall i \in \sigma_n} I_{SC}(i) & \text{otherwise} \end{cases}$$

where,

$input_{DFD}$: denotes $I_{DFD}(n), \ \forall n \in D_{DFD}$

$output_{DFD}$: denotes $O_{DFD}(n), \ \forall n \in D_{DFD}$

Definition 9.

- $f(DFD)$ denotes functions $f_{input} \bigcup f_{output}$

- $g(SC)$ denotes functions $g_{input} \bigcup g_{output}$

5. Identification of Design Discrepancies

5.1. Analysis.

Several interesting relationships between the designs may be investigated. Those that will contribute to the identification of design discrepancies are:

$$f(DFD) \sqsubseteq SC$$

$$g(SC) \sqsubseteq DFD$$

The transformation $f : DFD \longrightarrow SC$ maps the input/output relations of the Data Flow Design to a common abstract domain in which a comparison with the input/output relations of the Structure Chart Design can be performed. The same thing can be established for the transformation $g : SC \longrightarrow DFD$, where the input/output relations of the Structure Chart Design are mapped to a common abstract domain in order to be compared with the input/output relations of the Data Flow Design.

D1 is a Data Flow Design with input/output relations, denoted as DFD. *D2* is a Structure Chart design with SC as input/output relations. Assume that $f(DFD) \sqsubseteq SC$, the following issues are worth considering in the multiple view analysis phase.

- *(i)* Consider the cardinalities of their relevant nodes sets ((iii) and (iv) must hold):
  - if $|\text{R–Nodes}_{D1}| < |\text{R–Nodes}_{D2}|$ then some relevant nodes have been omitted in D1 or some extra relevant nodes have been added in D2.
  - if $|\text{R–Nodes}_{D1}| = |\text{R–Nodes}_{D2}|$ then *D1* and *D2* are graphically abstract designs.
- *(ii)* Consider the cardinalities of their relevant arcs sets ((iii) and (iv) must hold):
  - if $|\text{R–Arcs}_{D1}| < |\text{R–Arcs}_{D2}|$ then *D1* shows that some relevant data flows have been omitted or *D2* has some extra relevant data flows.
  - if $|\text{R–Arcs}_{D1}| = |\text{R–Arcs}_{D2}|$ then *D1* and *D2* are graphically abstract designs.
- *(iii)* Consider the labels from relevant nodes and relevant arcs:

  - if R–l–node$_{D1}$ $\subset$ R–l–node$_{D2}$ then labels of relevant nodes are not being preserved in *D1* or *D2*.
  - if R–l–node$_{D1}$ = R–l–node$_{D2}$ then labels from relevant nodes are being preserved, this is a necessary condition that has to hold between two graphically abstract designs.
  - if R–l–arc$_{D1}$ $\subset$ R–l–arc$_{D2}$ then labels from relevant arcs are not being preserved in *D1* or *D2*.
  - R–l–arc$_{D1}$ = R–l–arc$_{D2}$ then labels from relevant arcs are being preserved, this is a necessary condition that has to hold between two graphically abstract designs.

- *(iv)* Consider the roles from relevant nodes and relevant arcs:
  - if R–r–node$_{D1}$ $\subset$ R–r–node$_{D2}$ then roles from relevant nodes are not being preserved in *D1* or *D2*.
  - if R–r–node$_{D1}$ = R–r–node$_{D2}$ then roles from relevant nodes are being preserved, this is a necessary condition that has to hold between two graphically abstract designs.
  - if R–r–arc$_{D1}$ $\subset$ R–r–arc$_{D2}$ then roles from relevant arcs are not being preserved in *D1* or *D2*.
  - if R-r–arc$_{D1}$ = R–r–arc$_{D2}$ then roles from relevant arcs are being preserved, this is a necessary condition that has to hold between two graphically abstract designs.

The degree of composition has an upper limit which is determined by the loss of information from relevant nodes and arcs during the mapping process of functions *f* and *g*. Analyzing two graphically abstract designs, the function *f*, has an upper bound when *f*(DFD) is reached. At this point, the amount of information provided by the relevant nodes and arcs reaches a stable point, meaning that the composition *f*(DFD) is equal to SC. For the function *g* the stable point is reached at *g*(SC), meaning *g*(SC) is equal to DFD. Thus, for cases where the designs are not graphically abstract, it will be sufficient to compare at the same levels of composition since stable limits of relevant information are guaranteed to be present.

5.2. Types of Discrepancies

Incompatible information.

Identification of contradictory information in components (i.e., nodes, arcs) from different designs based on:

- Roles from relevant nodes are not being preserved.

  if R–r–node$_{D1}$ $\subset$ R–r–node$_{D2}$ then roles from relevant nodes are not being preserved in *D1* or *D2*.

- Roles from relevant arcs are not being preserved.

  if R–r–arc$_{D1}$ $\subset$ R–r–arc$_{D2}$ then roles from relevant arcs are not being preserved in *D1* or *D2*.

- Labels of relevant nodes are not being preserved.

  if R–l–node$_{D1}$ $\subset$ R–l–node$_{D2}$ then labels of relevant nodes are not being preserved in *D1* or *D2*.

- Labels of relevant arcs are not being preserved.

  if R–l–arc$_{D1}$ $\subset$ R–l–arc$_{D2}$ then labels from relevant arcs are not being preserved in *D1* or *D2*.

Missing information.

Missing design components identified with respect other design.

- Omission of relevant nodes.

  if |R–Nodes$_{D1}$| $<$ |R–Nodes$_{D2}$| then some relevant nodes have been omitted in D1 or some extra relevant nodes have been added in D2.

- Omission of relevant arcs.

  if |R–Arcs$_{D1}$| $<$ |R–Arcs$_{D2}$| then *D1* shows that some relevant data flows have been omitted or *D2* has some extra relevant data flows.

Inconsistency.

- Decomposition process is not being preserved.

$$\forall_{n \in Nodes_{D1},\, m \in Nodes_{D2}},\ n = m,\ \sigma_n = \sigma_m$$

- Input/Output relations of relevant nodes have not been preserved.

$$\forall_{n \in Nodes_{D1},\, m \in NodesD2},\ n = m,\ I_{D1}(n) = I_{D2}(m),\ O_{D1}(n) = O_{D2}(m)$$

6. An Illustrative Example

In this example, we illustrate the following steps in our MVA approach: (1) the formal encoding of the concrete designs; (2) the application of the abstraction function; (3) the identification of design discrepancies. The designs shown in figures 4, 5, and 6 are adapted from [17]. This test case presents inconsistency and missing information discrepancies. The designs are not graphically abstract with respect to each other. Note that diagram **D0** represent the high–level design; diagram **D6** represents the decomposition of node 6 in **D0**; and diagram **D12** represents the decomposition of node 12 in **D6**. Diagrams **D5** and **D11** are not shown.

6.1. *Concrete Graph Representation of the DFD Design*

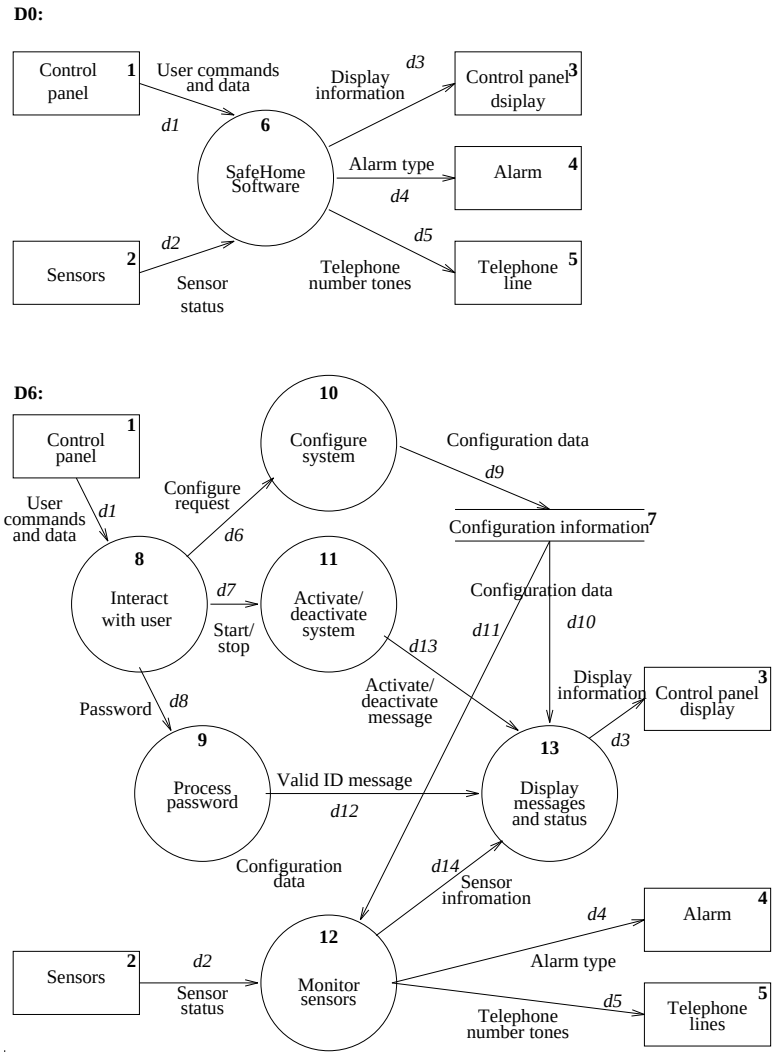

Figure 4: Part 1: Sample Data Flow Diagram.

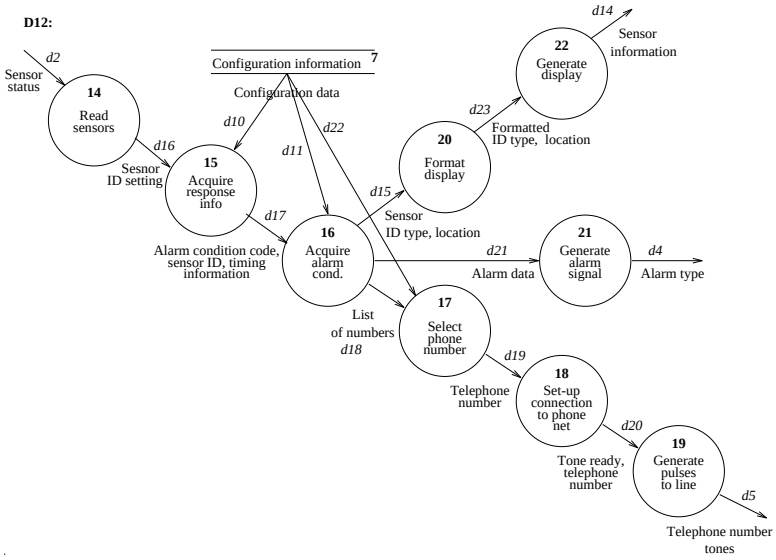

Figure 5: Part 2: Sample Data Flow Diagram.

$$
\begin{aligned}
\mathbf{Nodes_{D0}} &= \{ \quad 1,2,3,4,5,6\} \\
\mathbf{Nodes_{D6}} &= \{ \quad 1,2,4,5,7,8,9,10,11,12,13\} \\
\mathbf{Nodes_{D12}} &= \{ \quad 7,14,15,16,17,18,19,20,21,22\} \\
\sigma_{\mathbf{6}} &= \{ \quad 8,9,10,11,12,13\} \\
\sigma_{\mathbf{12}} &= \{ \quad 14,15,16,17,18,19,20,21,22\} \\
\pi &= \{\} \\
\lambda &= \{\} \\
\mathbf{arcs_{D0}} &= \{ \quad \langle 1,6\rangle , \langle 2,12\rangle , \langle 8,10\rangle , \langle 8,11\rangle , \langle 8,9\rangle , \\
&\qquad \langle 10,7\rangle , \langle 11,13\rangle , \langle 9,13\rangle , \langle 7,13\rangle , \langle 7,12\rangle , \\
&\qquad \langle 12,13\rangle , \langle 13,3\rangle , \langle 12,4\rangle , \langle 12,5\rangle \} \\
\mathbf{arcs_{D12}} &= \{ \quad \langle 2,14\rangle , \langle 14,15\rangle , \langle 15,16\rangle , \langle 16,17\rangle , \langle 17,18\rangle , \\
&\qquad \langle 18,19\rangle , \langle 19,5\rangle , \langle 7,15\rangle , \langle 7,16\rangle , \langle 7,17\rangle , \\
&\qquad \langle 16,20\rangle , \langle 16,21\rangle , \langle 20,22\rangle , \langle 22,13\rangle , \langle 21,4\rangle \} \\
\mathbf{arcs_{D5}} &= \{ \quad \langle 1,8\rangle , \langle 2,8\rangle , \langle 8,9\rangle , \langle 9,14\rangle \} \\
\mathbf{arcs_{D6}} &= \{ \quad \langle 9,14\rangle , \langle 14,10\rangle , \langle 14,11\rangle , \langle 10,15\rangle , \\
&\qquad \langle 11,15\rangle , \langle 15,7\rangle , \langle 7,4\rangle \} \\
\mathbf{arcs_{D11}} &= \{ \quad \langle 14,12\rangle , \langle 12,13\rangle , \langle 13,15\rangle \} \\
\mathbf{l\text{–}node} &= \{ \quad \langle 1, \text{Control-panel}\rangle , \langle 2, \text{Sensors}\rangle , \\
&\qquad \langle 3, \text{Control-panel-display}\rangle , \langle 4, \text{Alarm}\rangle , \\
&\qquad \langle 5, \text{Telephone-line}\rangle , \langle 6, \text{SafeHome-Software}\rangle , \\
&\qquad \langle 7, \text{Configuration-information}\rangle , \langle 8, \text{Interact-with-user}\rangle , \\
&\qquad \langle 9, \text{Process-password}\rangle , \langle 10, \text{Configure-system}\rangle , \\
&\qquad \langle 11, \text{Activate/deactivate-system}\rangle , \\
&\qquad \langle 13, \text{Display-messages-and-status}\rangle , \langle 14, \text{Read–sensors}\rangle , \\
&\qquad \langle 15, \text{Acquire-response-info}\rangle , \\
&\qquad \langle 16, \text{Acquire-alarm-cond}\rangle ,
\end{aligned}
$$

$\langle 17, \text{Select-phone-number}\rangle, \langle 12, \text{Monitor-sensors}\rangle,$
$\langle 18, \text{Set-up-connection-to-phone-net}\rangle,$
$\langle 19, \text{Generate-pulses-to-line}\rangle, \langle 20, \text{Format-display}\rangle,$
$\langle 21, \text{Generate-alarm-signal}\rangle, \langle 22, \text{Generate-display}\rangle\}$

l–arc $= \{$ $(\langle 1, 6\rangle, \{\text{User-commands-and-data}\}),$
$(\langle 2, 6\rangle, \{\text{Sensor-status}\}),$
$(\langle 6, 3\rangle, \{\text{Display-information}\}), (\langle 6, 4\rangle, \{\text{Alarm-type}\}),$
$(\langle 6, 5\rangle, \{\text{Telephone-number-tones}\}),$
$(\langle 1, 8\rangle, \{\text{User-commands-and-data}\}),$
$(\langle 2, 12\rangle, \{\text{Sensor-status}\}),$
$(\langle 8, 10\rangle, \{\text{Configure-request}\}),$
$(\langle 8, 11\rangle, \{\text{Start/stop}\}), (\langle 8, 9\rangle, \{\text{Password}\}),$
$(\langle 10, 7\rangle, \{\text{Configuration-data}\}),$
$(\langle 11, 13\rangle, \{\text{Activate/deactivate message}\}),$
$(\langle 9, 13\rangle, \{\text{Valid-ID-message}\}),$
$(\langle 7, 12\rangle, \{\text{Configuration-data}\}),$
$(\langle 7, 13\rangle, \{\text{Configuration-data}\}),$
$(\langle 12, 13\rangle, \{\text{Sensor-information}\}),$
$(\langle 12, 14\rangle, \{\text{Alarm-type}\}),$
$(\langle 12, 5\rangle, \{\text{Telephone-number-tones}\}),$
$(\langle 13, 3\rangle, \{\text{Display-information}\}),$
$(\langle 2, 14\rangle, \{\text{Sensor-status}\}),$
$(\langle 14, 15\rangle, \{\text{Sensor-ID-setting}\}),$
$(\langle 15, 16\rangle, \{\text{Alarm-cond-code, sensor-ID, timing-info}\}),$
$(\langle 17, 18\rangle, \{\text{Telephone-number}\}),$
$(\langle 16, 17\rangle, \{\text{List-of-numbers}\}),$
$(\langle 18, 19\rangle, \{\text{Generate-pulses-to-line}\}),$
$(\langle 7, 15\rangle, \{\text{Configuration-data}\}),$
$(\langle 7, 16\rangle, \{\text{Configuration-data}\}),$
$(\langle 7, 17\rangle, \{\text{Configuration-data}\}),$
$(\langle 16, 20\rangle, \{\text{Sensor-ID-type,location}\}),$
$(\langle 20, 22\rangle, \{\text{Formatted-Id-type,location}\}),$
$(\langle 22, 13\rangle, \{\text{Sensor-information}\}),$
$(\langle 16, 21\rangle, \{\text{Alarm-data}\}), (\langle 21, 4\rangle, \{\text{Alarm-type}\})\}$

id–arc $= \{$ $(\langle 1, 6\rangle, \{d1\}), (\langle 2, 6\rangle, \{d2\}), (\langle 6, 3\rangle, \{d3\}),$
$(\langle 6, 4\rangle, \{d4\}), (\langle 6, 5\rangle, \{d5\}), (\langle 1, 8\rangle, \{d1\}),$
$(\langle 2, 12\rangle, \{d2\}), (\langle 8, 10\rangle, \{d6\}), (\langle 8, 11\rangle, \{d7\}),$
$(\langle 8, 9\rangle, \{d8\}), (\langle 10, 7\rangle, \{d9\}), (\langle 11, 13\rangle, \{d13\}),$
$(\langle 9, 13\rangle, \{d12\}), (\langle 7, 12\rangle, \{d11\}), (\langle 7, 13\rangle, \{d10\}),$
$(\langle 12, 13\rangle, \{d14\}), (\langle 12, 4\rangle, \{d4\}), (\langle 12, 5\rangle, \{d5\}),$
$(\langle 2, 14\rangle, \{d2\}), (\langle 14, 15\rangle, \{d16\}), (\langle 15, 16\rangle, \{d17\}),$
$(\langle 16, 17\rangle, \{d18\}), (\langle 17, 18\rangle, \{d19\}), (\langle 18, 19\rangle, \{d20\}),$
$(\langle 7, 15\rangle, \{d10\}), (\langle 7, 16\rangle, \{d11\}), (\langle 7, 17\rangle, \{d22\}),$
$(\langle 16, 20\rangle, \{d15\}), (\langle 20, 22\rangle, \{d23\}), (\langle 22, 13\rangle, \{d14\}),$
$(\langle 16, 21\rangle, \{d21\}), (\langle 21, 4\rangle, \{d4\}), (\langle 13, 3\rangle, \{d3\})\}$

r–node $= \{$ $\langle 1, Entity\rangle, \langle 2, Entity\rangle, \langle 3, Entity\rangle, \langle 4, Entity\rangle,$
$\langle 5, Entity\rangle, \langle 6, Process\rangle, \langle 7, Store\rangle, \langle 8, Process\rangle,$
$\langle 9, Process\rangle, \langle 10, Process\rangle, \langle 11, Process\rangle,$
$\langle 12, Process\rangle, \langle 13, Process\rangle, \langle 14, Process\rangle,$
$\langle 15, Process\rangle, \langle 16, Process\rangle, \langle 17, Process\rangle,$
$\langle 18, Process\rangle, \langle 19, Process\rangle, \langle 20, Process\rangle,$

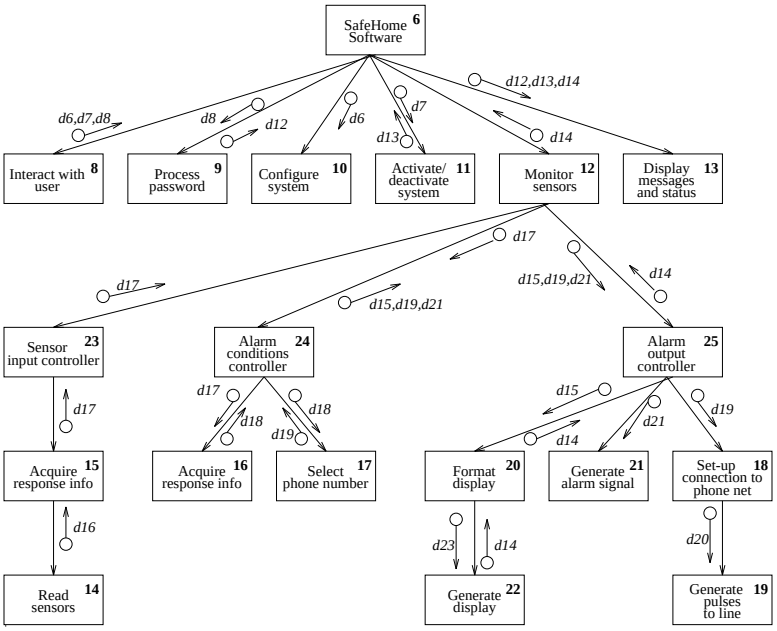

Figure 6: Sample Structure Chart Design.

$$
\begin{aligned}
\mathbf{r\text{-}arc} \quad = \{ \quad & \langle 21, Process\rangle, \langle 22, Process\rangle\} \\
& \langle d1, Dataflow\rangle, \langle d2, Dataflow\rangle, \langle d3, Dataflow\rangle, \\
& \langle d4, Dataflow\rangle, \langle d5, Dataflow\rangle, \langle d6, Dataflow\rangle, \\
& \langle d7, Dataflow\rangle, \langle d8, Dataflow\rangle, \langle d9, Dataflow\rangle, \\
& \langle d10, Dataflow\rangle, \langle d11, Dataflow\rangle, \langle d12, Dataflow\rangle, \\
& \langle d13, Dataflow\rangle, \langle d14, Dataflow\rangle, \langle d15, Dataflow\rangle, \\
& \langle d16, Dataflow\rangle, \langle d17, Dataflow\rangle, \langle d18, Dataflow\rangle, \\
& \langle d19, Dataflow\rangle, \langle d20, Dataflow\rangle, \langle d21, Dataflow\rangle, \\
& \langle d22, Dataflow\rangle, \langle d23, Dataflow\rangle\}
\end{aligned}
$$

6.2. *Concrete Graph Representation of the SC Design*

$$
\begin{aligned}
\mathbf{Nodes_{D0}} \quad &= \{ \quad 6, 8, 9, 10, 11, 12, 13, 14, 15, \\
& \qquad\quad 16, 17, 18, 19, 20, 21, 22, 23, 24, 25\} \\
\sigma_{\mathbf{6}} \quad &= \{ \quad 8, 9, 10, 11, 12, 13\} \\
\sigma_{\mathbf{12}} \quad &= \{ \quad 23, 24, 25\} \\
\sigma_{\mathbf{23}} \quad &= \{ \quad 15\} \\
\sigma_{\mathbf{24}} \quad &= \{ \quad 16, 17\} \\
\sigma_{\mathbf{25}} \quad &= \{ \quad 18, 20, 21\} \\
\sigma_{\mathbf{15}} \quad &= \{ \quad 14\} \\
\sigma_{\mathbf{20}} \quad &= \{ \quad 22\} \\
\sigma_{\mathbf{18}} \quad &= \{ \quad 19\} \\
\pi \quad &= \{ \quad \} \\
\lambda \quad &= \{ \quad \}
\end{aligned}
$$

$$
\begin{aligned}
\textbf{arcs} \quad &= \{ \quad \langle 8,6\rangle , \langle 6,9\rangle , \langle 9,6\rangle , \langle 6,10\rangle , \langle 6,11\rangle , \\
&\quad \langle 11,6\rangle , \langle 12,6\rangle , \langle 6,13\rangle , \langle 23,12\rangle , \langle 12,24\rangle , \\
&\quad \langle 24,12\rangle , \langle 12,25\rangle , \langle 25,12\rangle , \langle 15,23\rangle , \langle 14,15\rangle , \\
&\quad \langle 24,16\rangle , \langle 16,24\rangle , \langle 24,17\rangle , \langle 17,24\rangle , \langle 25,20\rangle , \\
&\quad \langle 20,25\rangle , \langle 25,21\rangle , \langle 25,18\rangle , \langle 20,22\rangle , \langle 22,20\rangle , \\
&\quad \langle 18,19\rangle \}
\end{aligned}
$$

$$
\begin{aligned}
\textbf{l–node} \quad &= \{ \quad \langle 6, \text{SafeHome-Software}\rangle , \\
&\quad \langle 8, \text{Interact-with-user}\rangle , \langle 9, \text{Process-password}\rangle , \\
&\quad \langle 10, \text{Configure-system}\rangle , \\
&\quad \langle 11, \text{Activate/deactivate-system}\rangle , \\
&\quad \langle 12, \text{Monitor-sensors}\rangle , \\
&\quad \langle 13, \text{Display-messages-and-status}\rangle , \\
&\quad \langle 14, \text{Read–sensors}\rangle , \\
&\quad \langle 15, \text{Acquire-response-info}\rangle , \\
&\quad \langle 16, \text{Acquire-alarm-cond}\rangle , \\
&\quad \langle 17, \text{Select-phone-number}\rangle , \\
&\quad \langle 18, \text{Set-up-connection-to-phone-net}\rangle , \\
&\quad \langle 19, \text{Generate-pulses-to-line}\rangle , \\
&\quad \langle 20, \text{Format-display}\rangle , \\
&\quad \langle 21, \text{Generate-alarm-signal}\rangle , \\
&\quad \langle 22, \text{Generate-display}\rangle , \\
&\quad \langle 23, \text{Sensor-input-controller}\rangle , \\
&\quad \langle 24, \text{Alarm-conditions-controller}\rangle , \\
&\quad \langle 25, \text{Alarm-output-controller}\rangle \}
\end{aligned}
$$

$$
\textbf{l–arc} \quad = \{(\quad \langle 8,6\rangle , \{\text{Configure-request,Start/stop,Password}\}) \ldots \}
$$

$$
\begin{aligned}
\textbf{id–arc} \quad &= \{ \quad (\langle 8,6\rangle , \{d6, d7, d8\}) \ldots \} \\
\textbf{r–node} \quad &= \{ \quad \langle 6, Module\rangle , \langle 8, Module\rangle , \\
&\quad \langle 9, Module\rangle , \langle 10, Module\rangle , \langle 11, Module\rangle , \\
&\quad \langle 12, Module\rangle , \langle 13, Module\rangle , \langle 14, Module\rangle , \\
&\quad \langle 15, Module\rangle , \langle 16, Module\rangle , \langle 17, Module\rangle , \\
&\quad \langle 18, Module\rangle , \langle 19, Module\rangle , \langle 20, Module\rangle , \\
&\quad \langle 21, Module\rangle , \langle 22, Module\rangle , \langle 23, Module\rangle , \\
&\quad \langle 24, Module\rangle , \langle 25, Module\rangle , \langle 23, Module\rangle \}
\end{aligned}
$$

$$
\begin{aligned}
\textbf{r–arc} \quad &= \{ \quad \langle d6, \text{Data-couple}\rangle , \langle d7, \text{Data-couple}\rangle , \\
&\quad \langle d8, \text{Data-couple}\rangle , \langle d9, \text{Data-couple}\rangle , \\
&\quad \langle d10, \text{Data-couple}\rangle , \langle d11, \text{Data-couple}\rangle , \\
&\quad \langle d12, \text{Data-couple}\rangle , \langle d13, \text{Data-couple}\rangle , \\
&\quad \langle d14, \text{Data-couple}\rangle , \langle d15, \text{Data-couple}\rangle , \\
&\quad \langle d16, \text{Data-couple}\rangle , \langle d17, \text{Data-couple}\rangle , \\
&\quad \langle d18, \text{Data-couple}\rangle , \langle d19, \text{Data-couple}\rangle , \\
&\quad \langle d20, \text{Data-couple}\rangle , \langle d21, \text{Data-couple}\rangle , \\
&\quad \langle d22, \text{Data-couple}\rangle , \langle d23, \text{Data-couple}\rangle \}
\end{aligned}
$$

6.3. *Abstract Graph Representation*

In table 1, f_{input}(n) and f_{output}(n) represent the input/output relations of an abstract SC. That is, the input/output relations (columns I_{DFD}(n) and O_{DFD}(n)) of a DFD (of figures 4 and 5), have been transformed into the input/output relations of its graphically abstract SC. Thus, I_{SC}(n), O_{SC}(n), f_{input}(n), and f_{output}(n) are

n	I_{DFD}(n)	O_{DFD}(n)	I_{SC}(n)	O_{SC}(n)	f_{input}(n)	f_{output}(n)	g_{input}(n)	g_{output}(n)
1	{}	{d1}	–	–	–	–	–	–
2	{}	{d2}	–	–	–	–	–	–
3	{d3}	{}	–	–	–	–	–	–
4	{d4}	{}	–	–	–	–	–	–
5	{d5}	{}	–	–	–	–	–	–
6	{d1,d2}	{d3,d4, d5}	{d6,d7, d8,d12, d13,d14}	{d6,d7, d8,d12, d13,d14}	{d6,d7, d8,d12, d13,d14}	{d6,d7, d8,d12, d13,d14}	{d6,d7, d8}	{}
7	{d9}	{d10,d11, d22}	–	–	–	–	–	–
8	{d1}	{d6,d7, d8}	{}	{d6,d7, d8}	{}	{d6,d7, d8}	{}	{d6,d7, d8}
9	{d8}	{d12}	{d8}	{d12}	{d8}	{d12}	{d8}	{d12}
10	{d6}	{d9}	{d6}	{}	{d6}	{}	{d6}	{}
11	{d7}	{d13}	{d7}	{d13}	{d7}	{d13}	{d7}	{d13}
12	{d2}	{d4,d5, d14}	{d14,d15, d17,d19, d21}	{d14,d15, d17,d19, d21}	{d14,d15, d16,d17, d18,d19, d20,d21, d23}	{d14,d15, d16,d17, d18,d19, d20,d21, d23}	{}	{}
13	{d10,d12, d13,d14}	{d3}	{d12,d13, d14}	{}	{d12,d13, d14}	{}	{d12,d13, d14}	{}
14	{d2}	{d16}	{}	{d16}	{}	{d16}	{}	{d16}
15	{d10,d16}	{d17}	{d16}	{d17}	{d16}	{d17}	{}	{d17}
16	{d11,d17}	{d15,d18, d21}	{d17}	{d15,d18, d21}	{d17}	{d15,d18, d21}	{d17}	{d15,d18, d21}
17	{d22,d18}	{d19}	{d18}	{d19}	{d18}	{d19}	{d18}	{d19}
18	{d19}	{d20}	{d19}	{d20}	{d19}	{d20}	{d19}	{}
19	{d20}	{d5}	{d20}	{}	{d20}	{}	{d20}	{}
20	{d15}	{d23}	{d14,d15}	{d14,d23}	{d15}	{d23}	{d15}	{d14}
21	{d21}	{d4}	{d21}	{}	{d21}	{}	{d21}	{}
22	{d23}	{d14}	{d23}	{d14}	{d23}	{d14}	{d23}	{d14}
23	–	–	{d17}	{d17}	–	–	{}	{d17}
24	–	–	{d15,d18, d19,d21}	{d15,d18} 19,d21}	–	–	{}	{d15,d19, d21}
25	–	–	{d14,d15, d19,d21}	{d14,d15, d19,d21}	–	–	{d15,d19, d21}	{}

Table 1: Input/Output Relations

in the same domain, a Structure Chart domain for further comparison.

Comparing columns I_{SC}(n) and O_{SC}(n) with columns f_{input}(n) and f_{output}(n) in table 1, we have the following:

- **Inconsistency in the decomposition process.** From the formalizations, we can identify that:

  Decomposition of the abstract SC denoted by f(DFD) (this follows from the decomposition of DFD):

$$\begin{aligned} \sigma_{\mathbf{6}} &= \{ \quad 8, 9, 10, 11, 12, 13\} \\ \sigma_{\mathbf{12}} &= \{ \quad 14, 15, 16, 17, 18, 19, 20, 21, 22\} \end{aligned}$$

  Decomposition of SC:

$$\begin{aligned} \sigma_{\mathbf{6}} &= \{ \quad 8, 9, 10, 11, 12, 13\} \\ \sigma_{\mathbf{12}} &= \{ \quad 23, 24, 25\} \\ \sigma_{\mathbf{23}} &= \{ \quad 15\} \\ \sigma_{\mathbf{24}} &= \{ \quad 16, 17\} \\ \sigma_{\mathbf{25}} &= \{ \quad 18, 20, 21\} \\ \sigma_{\mathbf{15}} &= \{ \quad 14\} \\ \sigma_{\mathbf{20}} &= \{ \quad 22\} \\ \sigma_{\mathbf{18}} &= \{ \quad 19\} \end{aligned}$$

- **I/O Relations for relevant nodes** 12 and 20 have not been preserved.
- **Omission of relevant nodes** 23, 24, 25 in Data Flow Diagram have been identified.

In table 1, g_{input}(n) and g_{output}(n) represent the input/output relations of an abstract DFD. That is, the input/output relations (columns I_{SC}(n) and O_{SC}(n)) of a DFD (figure 6), have been transformed into the input/output relations of its graphically abstract DFD. Thus, I_{DFD}(n), O_{DFD}(n), g_{input}(n), and g_{output}(n) are in the same domain, a Data Flow domain for further comparison.

In table 2, g_{input}(n) and g_{output}(n) represent $g(f(\text{DFD}))$ applied to columns f_{input}(n) f_{output}(n) of table 1. Comparing g_{input}(n) and g_{output}(n) of table 2 with columns I_{DFD}(n) and O_{DFD}(n) of table 1 we have the following:

Comparing columns g_{input}(n) and g_{output}(n) with columns I_{DFD}(n) and O_{DFD}(n) of table 1 we have the following:

- **Inconsistency in the decomposition process.** From the formalizations, we can identify that:

Decomposition of DFD:

$$\begin{aligned} \sigma_{\mathbf{6}} &= \{ \quad 8, 9, 10, 11, 12, 13\} \\ \sigma_{\mathbf{12}} &= \{ \quad 14, 15, 16, 17, 18, 19, 20, 21, 22\} \end{aligned}$$

Decomposition of the abstract DFD denoted by g(SC) (this follows from the decomposition of SC):

$$\begin{aligned} \sigma_{\mathbf{6}} &= \{ \quad 8, 9, 10, 11, 12, 13\} \\ \sigma_{\mathbf{12}} &= \{ \quad 23, 24, 25\} \\ \sigma_{\mathbf{23}} &= \{ \quad 15\} \\ \sigma_{\mathbf{24}} &= \{ \quad 16, 17\} \\ \sigma_{\mathbf{25}} &= \{ \quad 18, 20, 21\} \\ \sigma_{\mathbf{15}} &= \{ \quad 14\} \\ \sigma_{\mathbf{20}} &= \{ \quad 22\} \\ \sigma_{\mathbf{18}} &= \{ \quad 19\} \end{aligned}$$

- **I/O Relations for relevant nodes** 6,12,15,18,20 have not been preserved.
- **Omission of relevant nodes** 23,24,25 in Data Flow Diagram have been detected.

And from the comparison of their corresponding designs in figures 4, 5 and 7 we obtain the same conclusions.

Comparing columns g_{input}(n) and g_{output}(n) with columns I_{DFD}(n) and O_{DFD}(n) of table 1 we have the following:

- **The decomposition process is preserved.** From the formalizations, we can identify that:

  Decomposition of DFD:

$$\begin{aligned} \sigma_{\mathbf{6}} &= \{ \quad 8, 9, 10, 11, 12, 13\} \\ \sigma_{\mathbf{12}} &= \{ \quad 14, 15, 16, 17, 18, 19, 20, 21, 22\} \end{aligned}$$

  Decomposition of g(SC):

$$\begin{aligned} \sigma_{\mathbf{6}} &= \{ \quad 8, 9, 10, 11, 12, 13\} \\ \sigma_{\mathbf{12}} &= \{ \quad 14, 15, 16, 17, 18, 19, 20, 21, 22\} \end{aligned}$$

- **I/O Relations for all relevant nodes** have been preserved, considering the fact that some loss of information has occurred for nodes 6,8,10,12,13,14,15,16,17,19, and 21 due to the elimination of non-relevant inputs and outputs data flows to these nodes.

n	I_{SC}(n) (formerly f_{input}(n) in Table 1)	O_{SC}(n) (formerly f_{output}(n) in Table 1)	g_{input}(n)	g_{output}(n)
1	–	–	–	–
2	–	–	–	–
3	–	–	–	–
4	–	–	–	–
5	–	–	–	–
6	{d6,d7, d8,d12, d13,d14}	{d6,d7, d8,d12, d13,d14}	{}	{}
7	–	–	–	–
8	{}	{d6,d7, d8}	{}	{d6,d7, d8}
9	{d8}	{d12}	{d8}	{d12}
10	{d6}	{}	{d6}	{}
11	{d7}	{d13}	{d7}	{d13}
12	{d14,d15, d16,d17, d18,d19, d21,d23}	{d14,d15, d16,d17, d18,d19, d21,d23}	{}	{d14}
13	{d12,d13, d14}	{}	{d12,d13, d14}	{}
14	{}	{d16}	{}	{d16}
15	{d16}	{d17}	{d16}	{d17}
16	{d17}	{d15,d18, d21}	{d17}	{d15,d18, d21}
17	{d18}	{d19}	{d18}	{d19}
18	{d19}	{d20}	{d19}	{d20}
19	{d20}	{}	{d20}	{}
20	{d15}	{d23}	{d15}	{d23}
21	{d21}	{}	{d21}	{}
22	{d23}	{d14}	{d23}	{d14}

Table 2: Transformed Input/Output Relations

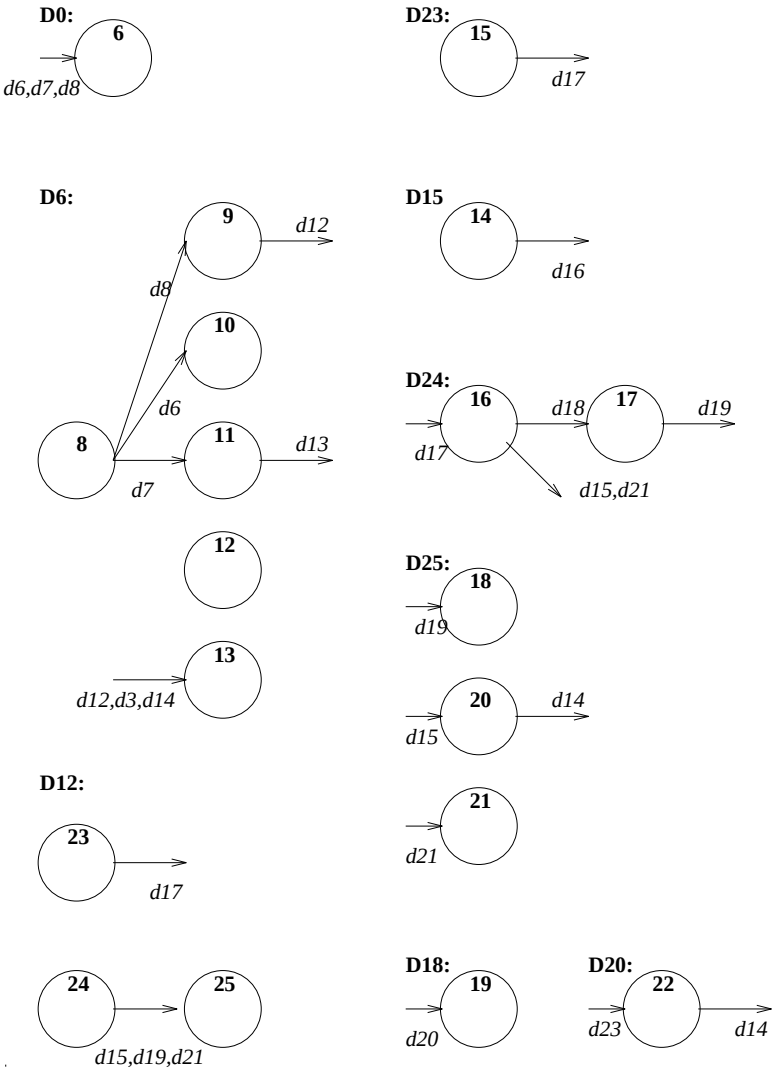

Figure 7: Data Flow Design obtained from columns g_{input}(n) and g_{output}(n) of table 2.

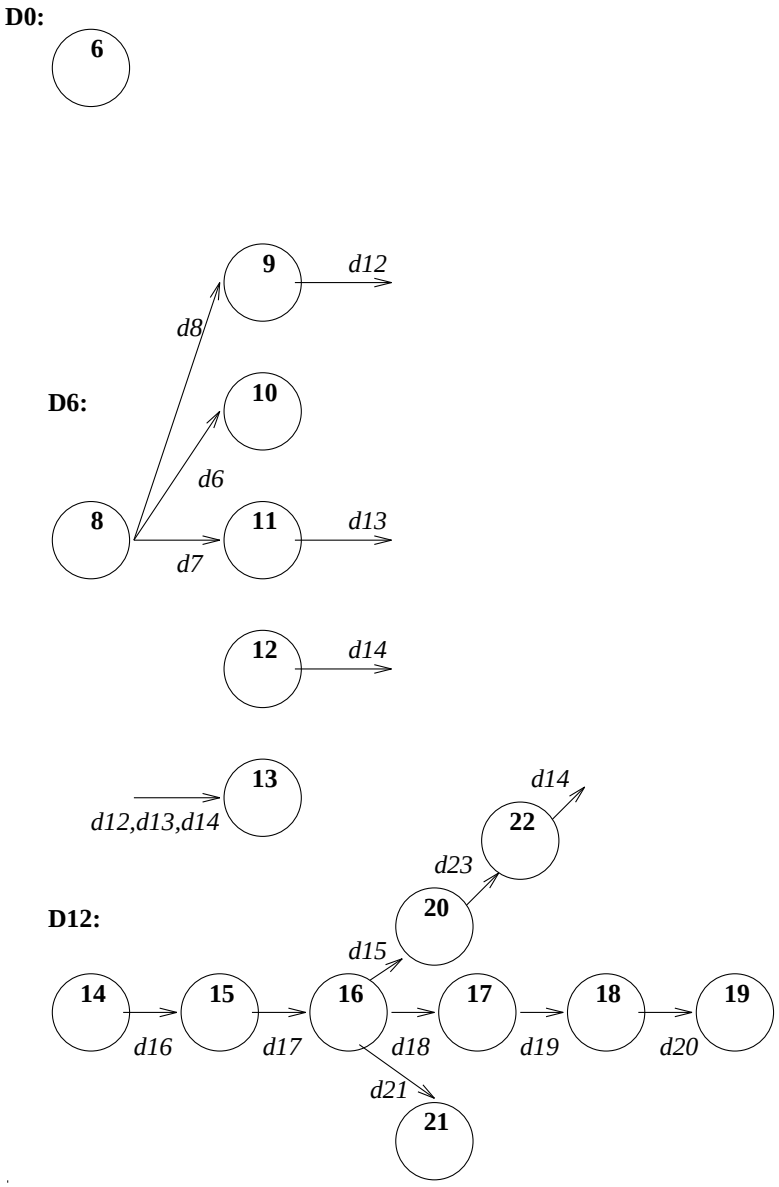

Figure 8: Data Flow Design obtained from columns g_{input}(n) and g_{output}(n) of table 2.

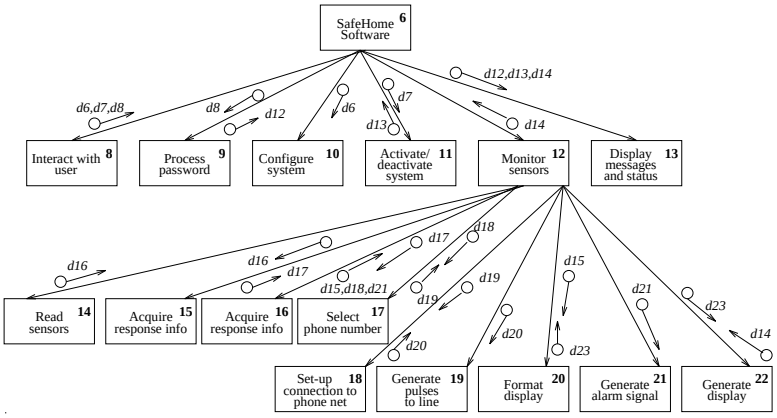

Figure 9: Structure Chart Design obtained from columns f_{input}(n) and f_{output}(n) of table 1.

- **Omission of relevant nodes** 23,24,25 in Data Flow Diagram have been detected.

And from the comparison of their corresponding designs in figures 4, 5 and 8 we can get to the same conclusions.

Figure 9 shows the structure chart design that is graphically abstract to data flow designs in figures 4 and 5. A comparison of the structure chart designs of figure 9 and figure 6 helps in the verification of the different decomposition processes.

7. Conclusions

During design formulation, designs are subjected to an evaluation and verification process, the goal of which is the detection of design errors. The analysis of different designs or "views" using different design notations has been proposed to achieve this goal. The formal support for the use of multiple views at the design stage with the purpose to detect design discrepancies was provided by formalizing the notation and the analysis process. Specifically in this work, a Data Flow Design and a Structure Chart Design capture a specific view of a system. Each view is formalized, independently analyzed, that is, each formalized view is checked for validity. The multiple view analysis, relying on the abstract interpretation framework, compares two abstract views, and identifies discrepancies between them. The type of design discrepancies identified were omission of information, missing information, and inconsistencies between views. Efficient use of information captured by multiple views represent a key issue in small scale design improvement process. Alternatively, the support that the proposed framework could provide in large complex systems design development might be considered (i.e. support of multiple "views" in software architecture).

References

1. D. Budgen, *Software Design*. International Computer Science Series, Addison-Wesley Publishing Co., 1994.
2. D. Marca, *SADT: Structured Analysis and Design Technique*. New York: McGraw Hill, 1988.
3. J. C. S. do Prado Leite and P. A. Freeman, "Requirements Validation Through Viewpoint Resolution," *IEEE Transactions on Software Engineering*, vol. 17, pp. 1253–1269, December 1991.
4. B. Nuseibeh, J. Kramer and A. Finkelstein, "A Framework for Expressing the Relations Between Multiple Views in Requirements Specification," *IEEE Transactions on Software Engineering*, pp. 760–773, October 1994.
5. S. P. Reiss, "PECAN: Program Development Systems that Support Multiple Views," *IEEE Transactions on Software Engineering*, vol. 11, pp. 276–285, March 1985.
6. M. R. Lowry and R. D. McCartney, *Chapter 5 in Automating Software Design*. Cambridge, England: MIT Press, 1991.
7. S. Easterbrook, "Domain Modeling with Hierarchies of Alternative Viewpoints," *IEEE International Symposium on Requirements Engineering*, pp. 65–72, January 4-6 1993.
8. S. D. Meyers, *Representing Software Systems in Multiple-View Development Environments*. PhD dissertation, Department of Computer Science at Brown University, May 1993. CS-93-18.
9. T. Tse, "The Application of Prolog to Structured Design," *Software Practice and Experience*, vol. 24, pp. 659–676, July 1994.
10. S. Hekmatpour and M. Woodman, "Formal Specification of Graphical Notations and Graphical Software tools," *Proceedings of 1st European Software Engineering Conference*, pp. 297–305, September 1987.
11. D. Harel, "On Visual Formalisms," *Communications of the ACM*, vol. 31, pp. 514–530, May 1988.
12. T. Tse and L. Pong, "Towards a Formal Foundation for DeMarco Data Flow Diagrams," *Computer Journal*, vol. 32, pp. 1–12, 1989.
13. Y. Tao and C. Kung, "Formal Definition and Verification of Data Flow Diagrams," *Journal of Systems Software*, vol. 16, pp. 29–36, 1991.
14. B. Ellington, "An Automated System for Multiple Views Analysis," Master's thesis, Department of Computer Science at Tulane University, August 1998.
15. C. Lemus-Olalde, *Abstract Interpretation of Multiple Views in Software Design*. PhD dissertation, Department of Computer Science at Tulane University, May 1996.
16. P. Cousot, and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction of Approximation of Fixpoints," *Proceedings of the 4th ACM Symposium Principles of Programming Languages*, pp. 238–252, 1977.
17. R. Pressman, *Software Engineering A Practioner's Approach*. McGraw-Hill, third edition ed., 1992.