Sci. J. Fac. Sci., Minufiya Univ., Vol. XIV,2000, 259-282

Semantically Extended Data Flow Diagram As A formal Specification Tool for Methodology Information Exchange Service (MIES Problem).

*** Fatma A. Omara**
**** Reham A. Mahmoud**

* Ass. Prof. In Computer Science & Eng. Dept., Faculty of Electronic Eng. Menouf, Minufiya University.
** Research Student, Mathematics Dept., Faculty of Science, Minufiya University.

ABSTRACT

In this paper a method for associating the Data Flow Diagram (DFD) with a formal specification is described. The intention is to enhance the using of the DFD as a formal specification tool, and to gain a tool that can be used to document the application functionality in an understandable manner and, at the same time, be capable of producing a formal specification that can be used to rigorously investigate semantic properties of the application. The formal specification technique is used here based on the algebraic specification technique.

1. INTRODUCTION

A framework for semantically extending an informal specification tool is structured. Namely, the Data Flow Diagram (DFD) of Structured Analysis (SA) (See Appendix –A) [1]. The framework facilitates the use of the DFD as an effective documentation tool, as well as a formal specification tool. A formal view can be used to support rigorous verification and investigation of the application properties. A DFD models an application in terms of the following elements [2]:

a) ***Data-Transforms:*** Abstractions of an application's processing elements.

b) ***Data-Stores:*** Abstraction of repositories of data.

c) ***External-Entities:*** Abstraction of external objects interacting with the application.

d) ***Data Flows:*** Abstractions of data communication between the above elements.

2. Semantically Extended DFD's : An Overview

A Control–Extended DFD (C-DFD) is a DFD supplemented with notation for describing control dependencies among its elements [2][3]. A Semantically Extended DFD (EXT–DFD) is defined as a Control Extended DFD associated with formal semantics. EXT–DFD has two aspects: Syntactic and Semantic; The Syntactic aspect of an EXT-DFD is concerned with the pictorial representation, which is C-DFD of the application. According to the work here, the semantic aspect of an EXT-DFD is concerned with the behavioral interpretation of its C-DFD [2].

The specification characterizing the semantics of a C-DFD is called its Behavioral Specification (BS). An EXT-DFD can be viewed as C-DFD and its BS (See Appendix-A).

Semantic Aspects of the EXT-DFD constructs as follows (see Fig. 1):-

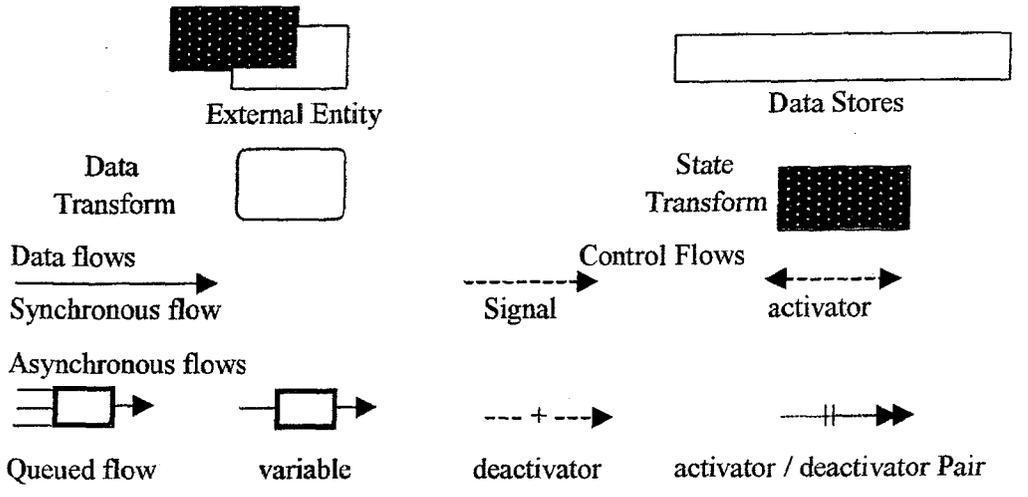

Fig. 1 Basic EXT-DFD Constructs.

The data and state transforms, asynchronous flows, external entities, and data stores of a C-DFD are called its components.

C-DFD components are semantically interpreted as processes, and a C-DFD is interpreted as a system of communicating process (a system can also be viewed as a single processes which is considered as an entity associated with a set of states and events and a class of behaviors). Intuitively, a behavior is an execution, of a process. During a process execution events occur, where such occurrences are called actions. At any point in an execution, a process is in a single, possibly composite state. Actions cause certain states change to other states, such changes are called (state) transitions [4].

A process behaviour is formally defined as a labeled sequence of transitions, $S_1 \xrightarrow{L_1} S2 \xrightarrow{L_2} \ldots \xrightarrow{L_{n-1}} Sn$, where the label, L_i is the action causes state S changes to S, and $\mathbf{1} \leq i \leq (n-1)$.

This formula represents the effect of a communication action. This action associated with communication via a process data. The labeled transition $S \xrightarrow{L} S'$ is formally interpreted as follows: the action represented by the label L_i causes the state S changes to state. S'. The class of behaviors associated with a process is defined by a labeled state transition system, $< S, L, T >$, where S is a set of process states, L is a set of labels representing communication actions, and T is a

relation [S, L, S], whose elements are labeled transitions. In this work an algebraic approach is used to specify a labeled state transition system. Processes are viewed as Abstract Data Types (ADT's) and are specified algebraically by Algebraic State Transition System (ASTS's), as follow [5][6] :-

Transition Specification is Spec Name

State Specification is State Spec

Label Specification is Label Spec

Transition relation is state $\xrightarrow{\text{Lable}}$ **state: state label state**

(a) The State Specification (State Spec):

It is an algebraic specification characterizing the states of the process. It defines state constructors, where its function creates process states [7].

(b) The Label Specification (Label Spec):

It is an algebraic specification characterizing the labels of the process. It defines label constructor, that is, its functions creates process labels. The label constructor PAR is defined in all label specifications with functionality

PAR : Set label ⟶ label,

Where, a set label instance is a set of labels of type label, the constructor PAR takes a set of labels, and returns a label representing the parallel occurrences of the actions represented in the argument label set as follow [8]:-

The axiom characterizing PAR in label spec is :

For all Lset, S : Set label

PAR ({ PAR (Lset) } U S) = PAR (Lset U S)

Where, { PAR (Lset)} is a singleton

And U is the set union operation.

(c) The Transition Axioms:

Axioms of an Algebraic State Transition System (ASTS) define the transitions of the process's state transition system. The axioms, are of the form:

$$C \Longrightarrow S_i \xrightarrow{L} S_j$$

Where S_i, S_j are states defined in state spec, L is a label defined in label spec, and C is a boolean expression stating the conditions under which the transition in the conclusion can take place; that is, if C is true, then the transition $S_i \xrightarrow{L} S_j$ can take place. If a transition can't be shown to be true in an ASTS's axiom system, then it is said to be undefined in the ASTS [9]. The semantics of a C-DFD are

characterized by an ASTS created in a bottom-up manner from ASTS's characterizing the semantics of individual C-DFD components in the following manner:

(1) Derive ASTS's characterizing the behaviour of each C-DFD component from specifier–supplied descriptions. The resulting set of ASTS's, together with the C-DFD, is called the Basic Interpreted C-DFD [4].

(2) Derive an ASTS characterizing the synchronous interactions that can take place among C-DFD components from the Basic Interpreted C-DFD. This ASTS is called the Synchronous Interaction Specification (SIS). The SIS, together with the C-DFD, is called the Basic Ext–DFD [2].

(3) Derive an ASTS characterizing the permissible time–dependent relationships among the synchronous interactions specified in the SIS from the Basic Ext–DFD. The resulting ASTS is called the Behavioral Specification (BS). A DFD representation of the specification – generation method is given in Fig 2:-

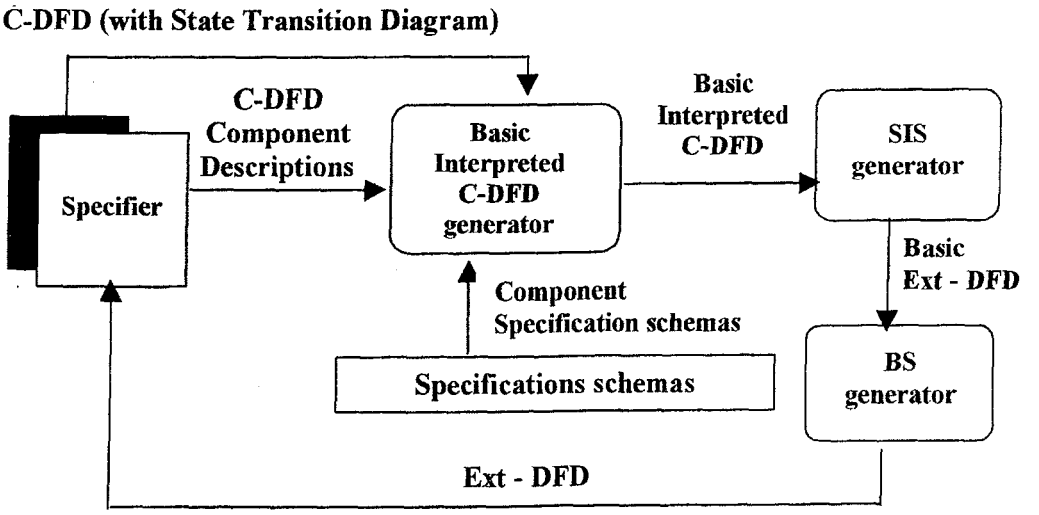

Fig.2 Extracting the Formal Specifications from C-DFD'S

(I) Phase one of the Specification–Generation Method:-

ASTS'S characterizing semantic models of C-DFD components are created from specifier–supplied descriptions and specification schemas. The specifier–supplied descriptions are analogous to the traditional Structured Analysis (SA) data dictionary definitions and data transform specifications[8]. For (asynchronous) data flow and data store components, specification schemas are instantiated with formal specifications derived from the specifier's descriptions of transmitted or stored data to produce formal specifications characterizing their structure and data access behavior.

(II) Phase two of the specification:-

It determines which process actions defined in phase one are to be synchronized. For example, a read from a data store action of a data transform must be synchronized with a read action the data store. A specification defines the effects of synchronous transitions. That is, transitions caused by synchronizing actions is derived form the data and control relationships depicted in the C-DFD using rules which state what type of interactions depicted in a C-DFD result in synchronous transitions [10].

The driven SIS in phase two is:

The ASTS defines the synchronous interaction that can take place among the processes defined in phase one and is of the form :

Synchronous Interaction Specification is Synchspec

C-DFD Component Specifications are Compspec

Application State Specification is Appstate

Application Label Specification is Applabel

Synchronous transition relation S $\xrightarrow[\text{Label}]{\text{Application}}$ **S' : appstate applabel appstate**

The C-DFD component specifications named in compspec are the ASTS'S generated in phase one. The application state type, appstate, is an aggregation of its C-DFD component state types and is algebraically defined in appstate [11].

The specification Applabel defines the SIS label constructor **SYNCH**, with functionality **SYNCH** :

Set label ⟶ applabel

Where instances of set label are sets of component labels specified in the Basic Interpreted C-DFD. The constructor **SYNCH** takes a set of component labels and returns a single label representing the action formed by synchronizing the actions represented by the argument label set. The resulting label is called a synchronous label and is denoted **SYNCH** (L_1, ... L_n) where L_1 ..., L_n are the labels in the argument set of Labels, the action represented by SIS labels are called synchronous actions [12].

The axioms of the SIS, Synch Axioms, define transitions caused by synchronous actions, for example:-

Given that the transitions $P_1 \xrightarrow{L_1} P'_1$, ... $P_J \xrightarrow{L_J} P'_J$. Can take place in an application state $< P_1, \ldots\ldots, P_j, P_{j+1}, \ldots P_n >$ then the synchronized effect of the actions labeled $L_1, \ldots, L_j$ is defined by axiom :

$$P_1 \xrightarrow{L_1} P'_1, \ldots P_J \xrightarrow{L_J} P'_J, L = SYNCH(L_1, \ldots, L_j) \Longrightarrow$$
$$< P_1, \ldots, P_j, P_{j+1}, \ldots P_n > \xrightarrow{L} < P_1, \ldots, P_j, P_{j+1}, \ldots P_n >$$

(III) Phase Three of the Specification:-

In this phase, constraints on when the synchronous interactions can take place are specified. The ASTS is used to derive the Behavioral Specification (BS) which is considered as an extension of the SIS. The BS concerns definitions of the effects of actions represented by parallel action labels (application labels built using the PAR constructor) [13][14].

3. Semantically Extended-DFD of Methodology Information Exchange Service (MIES) Problem:

The MIES problem in the Multimedia environment, is concerned with transmitting the pictures, voices; especially pictures the form of stream of frames using Open Distributed Processing systems (ODP) (See Appendix-A) [15]. According to MIES problem three processes are used: Generating stream of frames for pictures, Monitoring stream of frames for pictures according to Quality of Service Violation (QoS-Violation) process and finally the Receiving stream of frames for pictures. The formalization of this problem is driven by using the Ext-DFD [16]. The Monitoring state of pictures which belongs to QoS-Violation phase will be explained in details. The C-DFD elements of the Monitoring State of pictures are be illustrated in Fig. 3. When the producer generates the stream of frames, they will be sent to the check stream. The check stream state examines the stream of frames by putting this stream in a monitoring mode (Monmod) to check if it is abnormal or normal according to the Quality of Service conditions. If it is abnormal then check stream state goes toward the modify stream state via monitor and runs the modifications. According to their priority of arrivals of the (Rt state 1) which is the state of the input stream which will be modified and after modifying the state of the output stream (Wt state), then the final modified input state (Rt state 2) will directly goes toward the check stream state and becomes stream status (Sstatus) then arrive to the consumer. The Generating stream phase can be implemented exactly as Monitoring State; but with no consumer and Modify Stream state, and Generate Stream is replaced by Check stream. Also, the Receiving phase will be implemented in the same way with replacing Check stream state by the stream state, and replacing Receive stream by the Check stream.

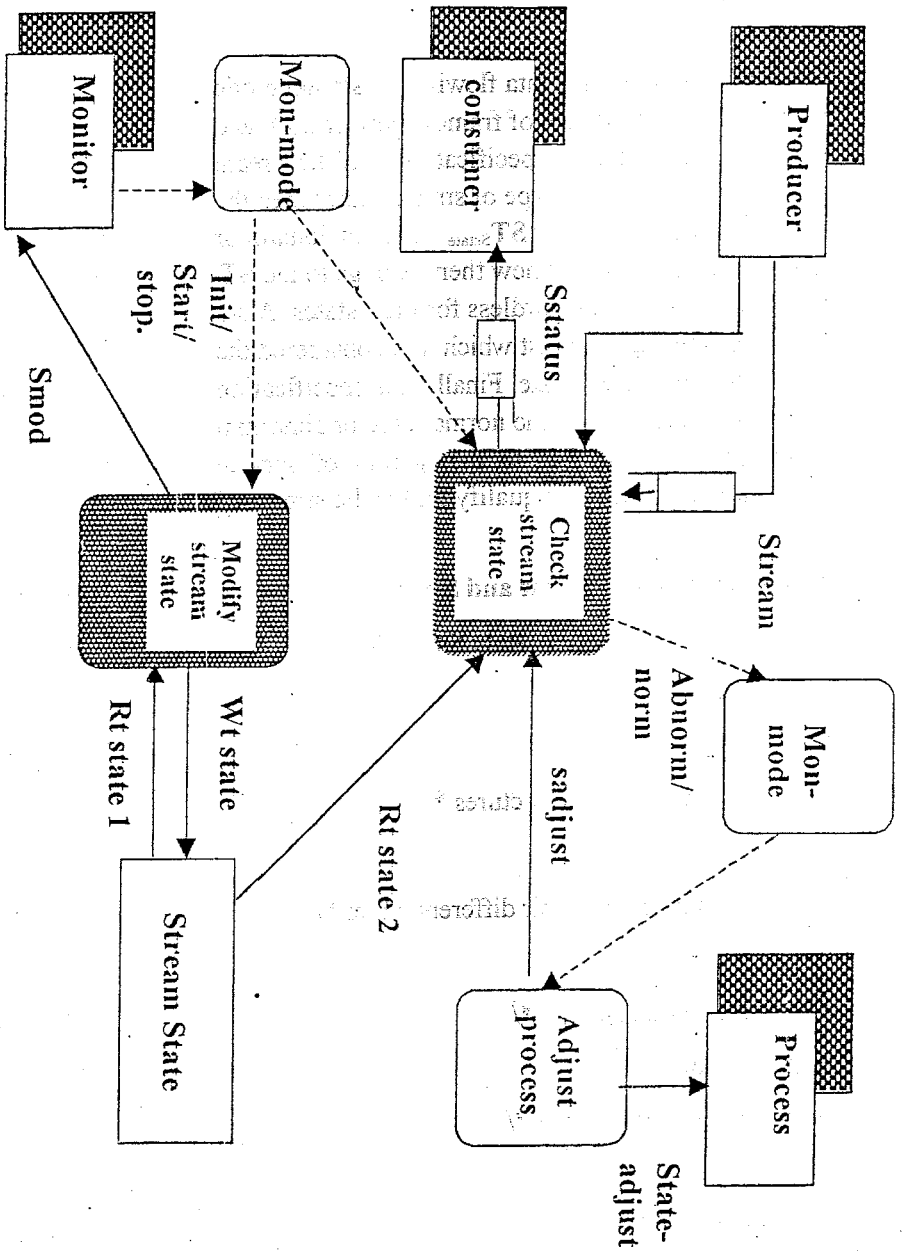

Fig. 3: C-DFD for Monitoring state of MIES Problem.

3.1 The Basic Interpreted C-DFD of Monitoring Process for MIES problem:

3.1.1. Phase one

In this phase, specifier created descriptions of C-DFD components are used to creat ASTS's characterizing their behavior [2][15].

A- Specifying the Data Domain:

The data transmitted on a C-DFD's data flows and stored in its data stores make up the data domain of the C-DFD. A C-DFD'S data domain is defined by the specifier using a Data Description (DD) language. A DD language provides functions for constructing data types from elementary (non decomposable) data types. Elementary types are associated with algebraic specification that define functions on the instances. Two kinds of constructors are used in a DD to build more complex types from elementary types; Aggregate and Union constructors. The DD formats corresponding to the type constructors are :

$$\textbf{Aggregate : } t = < t_1, \ldots\ldots, t_n >$$

The type t is an aggregation of the data types $t_1, \ldots, t_n$. Each type t_i, $i=1, \ldots. n$, is associated with a unique identifier which is the type name, t_i, if the name is unique, or a unique alias if the type name is not unique in the aggregate.

$$\textbf{Union : } t = t_1 + t_2$$

The type t can either be t_1 or t_2. This type is associated with boolean functions of the form is_t where t is a type in the union and is_t **(d)** is true if and only if d is of type t.

Specification 1 illustrates the data domain for the MIES problem which is considered the stream of frames for pictures. Also it defines the data stored in each stream state which could be stream state of frames for past picture encoded as i-pic or for current picture has closest past picture encoded as b-pic.

Specification 1: Date Domain (DD) defining the data stored in stream state.

Data description is Sstate

/* Data domains is stream states of frames for pictures */

Sstate = < i – Pic, p-Pic, b-Pic >

/*stream states are states of past pictures, current picture of future pictures */

Where for all Sm : Sstate;

The identifier to define

i – pic = i - encoded (frame) /* stream state of past picture */

p – pic = (frame, closest past) /* stream state of current picture */

b-pic = (frame, closest past, closest future)

/* stream state of future picture */

Specification 2 explains the data domain for all data flowing on stream mode (Smod) which it is a state illustrates the kind of stream of frames (new or not), and the kind of its state according to the previous states in specification 1. If the stream of frames is new according to comm which is a union type of stream states then the boolean function is_{Sstate} is true, and the state S is a type of ST_{Sstate} which is domain of all stream of frames and their states. If the stream isn't new then belongs to the $ST^{\backslash}$ which it is the domain of the streams of frames with regardless for their states. Also, this specification illustrates the domain of the state Sadjust which it is considered the adjust stream of frames for picture and its current state. Finally this specification explains the domain of the state Sstatus which shown the normal case or abnormal case for the stream of frames for picture according to the quality of service conditions through the union type Sindic expresses the qualifying for the stream of frames for pictures.

Specification 2: DD's for data flowing on Smod, Sadjust and Sstatus.

Data description is Smod /* data description for the state Smod */

Using Sstate /* using stream states */

Comm = New + i – pic + p-pic + b – pic

/* domain illustrates the different states of the frames for pictures */

ST = Stream + Sstate

/* type of elements contains the different streams with their different states */

$ST^{\backslash}$ = Stream

/* the type of elements contains the different streams only */

Smod = < Comm, S > Where for all S : Smod

/* the aggregation domain includes the stream and its state */

1. S. comm = New $\Longrightarrow$ is_{Sstate} (S.ST)

/* if the stream is new then belongs to the type ST */

2. S. comm ≠ New $\Longrightarrow$ $is_{Sstream}$ ($S.ST^{\backslash}$)

/* if the stream isn't new then belongs to type $ST^{\backslash}$ */

The type Smod is a two- place aggregate in which the first place indicates the type of picture range modification required. If the first position has value New then the range in the data store stream state is to be replaced by the range in the aggregates second position remains as a previous state of the Smod aggregate.

Data description is Sadjust

/* data description of the adjust stream and its state */

using Sstate

Sadjust = < picture : streams Sstate >

The type Sadjust is an aggregate in which the first position identified by picture is a picture value and the second position is picture state.

Data description is Sstatus /* data description of the stream status */

Sindic = Normal + Abnormal

/* union type of normal and abnormal cases of the stream */

Sstatus = < pictures streams, Sindic >

/* stream of frames according to the QoS conditions */

The type Sstatus is an aggregate in which the first position (identified by picture) is a picture value and the second position is a value indicating whether the picture is within or out of the normal range.

Specification 3: data specification for the data flowing on Smod and Sstatus

Data specification is Smod /* data specificaiton for streams mode */

Using Sstate /* using stream states */

Sorts Smod, Comm, ST

/* the sorts used to build this specificaiton is stream mode elements, comm elements and ST elements */

Constructors

New, i – pic, p – pic : S ⟶ Comm

/* the stream may be has different states and so, towards to comm type */

_: Sstate ⟶ ST

/* the stream may be only one stream of one frame in ST domain */

< _> : Stream ⟶ ST

/* the stream state may be the aggregation of only one stream of many frames in ST domain */

< _, _>: Comm, ST ⟶ Smod

/* the aggregation domain all its elements have the properties of the domains Comm & ST and these elements is a type of Smod state */

Definition of the predicate

Ok_{Smod} : Smod /* state belongs to Smod state */

Axioms

For all S1, S'1 : C: Comm

1. Ok_{Smod} (< New, < S1, S'1 >>)

/* if Ok_{Smod} state is new then the stream S1 changes to stream S'1 */

2. C ≠ New ⟶ Ok_{Smod} (<C, < S1>>)

/* if the Ok_{Smod} state isn't new then the stream has the same state */

Data specification is Sstatus

Sorts Sstatus, Sindic

Constructors

Normal, Abnormal : ⟶ Sindic

< _, _ > : Stream Sindic ⟶ Sstatus

/* the aggregation domain, all its elements have the properties of the domains Sindic and these elements is a type of Sstatus state */

Constraints on data types are defined using a where statement which is of the form where P, where P is the first – order predicate formula. For example, if the first and last position in an aggregate type must maintain a less than or equal to relationship, this is expressed as:

$t = < t_1, ..., t_n >$ where for all a : t, $(a.t_1) \leq (a.\ t_n)$

/* a type t_n contains the data transmitted and stored in the type t_n */

A DD is of the form:

Data description is DD name

/* description of the data domains name of the type */

Using DDes

/* using the data domains for same of the data transmitted and stored */

Type Defns

/* explaining the arithmetic operations on rational numbers */

Where Data Domains names (DDes) are the names of DD's defining the types that are used to define the types in type Defns. Specification 1 and Specification 2 give DD's for some of the data transmitted and stored in the C-DFD (See

Appendix –A). The type number is elementary in the used DD language, where an instance of number is a rational number which is associated with the usual arithmetic operations on rational numbers. The data flow pictures transmits data of type number- that is, picture states are represented by numbers. The data flows (rtstate1), (rtstate2) and (wt state) transmit data of type state (see in detail Refs. [2], [15], [16]). Similarly the first specification of generating and receiving stream of frames for defining data domain for these processes can be done.

B- Specifying Data Flows:

A data flow is interpreted as either an asynchronous or synchronous data interface between its generator and receivers. A synchronous data flow is one which requires its generator and receivers to cooperate in order for data communication to take place. Communication via a synchronous flow is effective only when all the receivers and the generator of the flow are ready to accept and send data, respectively, on the flow. The behavior of asynchronous data flow is implicit in the specification of the synchronous interactions between its generator and receivers, which is created in the second phase of our specification generation process [2].

Specification 4: state and label specification for the Queued flow Sstatus.

Queued flow State Specification is QS–Sstatus

/* the states are arranged in queue structure */

Data specification is Sstatus

State is Queue

Constructions

Empty q : ⟶ Queue /* if there is empty queue, then no elements*/

Add q : Sstatus queue ⟶ Queue

/* if the operation is adding then the queue increase with new stream statue (Sstatus) */

Operations

Del q: Queue ⟶ Queue

/* del operation means remove element from the queue */

Top : Queue ⟶ Sstatus

/* top operation means add a new stream status */

Operation axioms

For all e : Sstatus; q : Queue

1- **del q (add q, (e, empty q)) = empty q**

/* del operation of the empty queue returns to the empty queue */

2- **q ≠ empty q = del q (add q (e,q)) = add q (e, del q (q))**

/* del operation of the element has already added so, no effects */

3- **q ≠ empty q = top (add q (e, q)) = top (q)**

/* top operation means the increasing of the elements in queue */

4- **top ((add q) (e, empty q)) = e**

/* add operation to the empty queue so, the queue will be increased by one element */

Queued flow Labels Specification is QL – Sstatus

/* the specification of the labels */

Data specification is Sstatus

Label is qlable

Constructors

$Send_{Sstatus}$, $Receive_{Sstatus}$: Sstatus ⟶ qlabel

/* lables domains which arrange in queue have the operations send, receive */

<u>Specification 5</u>: Queued flow specification for data flow Sstatus

Transition specification is Q – Sstatus

State specification is QS – Sstatus

Label specification is QL – Sstatus

Transition relation is q $\xrightarrow{L}$ q : Queue qlable Queue

Transition Axioms

For all Sstatus, q : Queue, L : qlabel

1- **L = $Receive_{Sstatus}$, (e) ⟶ q $\xrightarrow{L}$ add q (e, q)**

/* receive operation means adding the new element e to the queue */

2- **q ≠empty q, L = $Send_{Sstatus}$ (top (q)) ⟶ q $\xrightarrow{L}$ del q (q)**

/* send operation means removing an element e from the queue */

The ASTS characterizing the behavior of a queued flow is created by instantiating the Queued Flow Transition Schema (QFTS) with the Queued flow's state and label specifications, while the ASTS for a variable is generated by instantiating the Variable Transition Schema (VTS) with variable state and label specification. An instantiation of the QFTS for the data flow Sstatus is given in

specification 5. Axioms 1 and 2 of Q-Sstatus define the change in states caused by the receive and send actions, respectively.

C- Specifying Data Stores

In the process vie of a data store, a data store state is an instance of an abstract data type (e.g., an indexed list of values), and a data store label represents a data store access action. Data store transitions define the effects of access actions on data store states [2], [15].

Specification 6: **A single value data store DSS.**

Data store state specification is Name

Data specification is data spec (sort is data)

Sorts readvalue, store

/* the sorts of the operation are used read, store */

constructors

<_>: data ⟶ readvalue

/* the read operation means increasing element in the data store */

Undef: ⟶ readvalue

/* the read operation may be a reading to undefined value */

empty store: ⟶ store

/* the empty store can't be a domain for any operation */

put: data ⟶ store

/* the put operation means adding element to the store */

Access operations

Read : store ⟶ readvalue

- **Reads the value in the data store.**
- **If data store is empty it returns the value undefined.**

Writ: store data ⟶ store

- **Replaces the value in the data store with a new value**

/* the general definition of the write operation */

Axioms

for all S : store; V : data

1- **read (empty store) = undef**

/* the read operation of the empty store returns the undefined value */

2- **read (put (V)) = <V>** /* the read operation of the value V*/

3- **write (S,V) = (V)** /* the write operation of the value V*/

Specification 7: ASTS for data store stream state

Transition Specification is Sstate

/* transition specification of stream state */

State Specification is Sstate' /* state specification of stream state */

Label Specification is Sstate label

/* label specification of stream state */

Transition relation is S1 $\xrightarrow{L}$ **S1' : Spstate Splabel Spstate**

Transition Axioms

For all d : Sstate, N : Spstate, L : Splabel

1- L = Read ('rt state 1', d), read (N) = <d> $\Longrightarrow$ **N** $\xrightarrow{L}$ **N**

/* the read operation to the input stream state (rt state 1) which is a type of d*/

2- L = Read ('rt state 2', d), read (N) = <d> $\Longrightarrow$ **N** $\xrightarrow{L}$ **N**

/* the read operation to the final modified input stream state (rt state 2) which is a type of d*/

3- L = write ('wt state', d) $\Longrightarrow$ **N** $\xrightarrow{L}$ **Put (d)**

/* the write operation to the output stream state (wt state) which is a type of d */

D. Specifying Data Transforms

A data transform is an entity that transforms input data to output data and/or signals. The outputs of a data transform depend solely on its current inputs. This isn't a serious restriction on what can be modeled by a data transform, since processing elements whose outputs are also dependent on past inputs can be modeled as a data transforms communication with a data store that stores the effects of past executions. The behavior of a data transform is defined by the specifier via a Data Transform Specification (DTS). DTS consists of two parts: a header and a statement. The header of a DTS declares the name of the DTS and the variables used in the DTS's statement part. The statement language consists of a set of basic statements and a set of statement constructors.

Specification 8: data transform specification for modify stream state.

DTS modify stream state (in 1: Smod; in 2: Sstate)

/* data transform specification for modify stream state according to value in 1 which belongs to stream mode domain and the value in 2 which belongs to stream state domain */

Receive ('Smod', in 1);

/* receive operation for the element in 1 which belongs to Smod */

If (in 1. Comm ≠ new) then

/* If in 1 state which belongs to comm domain not new then send the input stream state (rt state 1) which belongs to Sstate */

Send ('rt state 1', in 2);

If (in 1. Comm = i – pic) then

/* If in 1 state which belongs to comm domain is the state for stream of frames for past picture then receive to (wt state) as output stream and in 1 state is type of St domain and in 2 state is the state for stream of frames for future picture */

Receive ('wt state' , < in 1. St, in 2. b-pic>)

Else

Receive ('wt state', < in2. b- pic, in 1. St>

/* If the previous condition not true then receive to wt state as output stream and in 2 state is the state for stream of frames for future picture and in 1 state is type of St domain */

Else

Receive ('wt state', in 1. St);

E. External Entities

In structured analysis, descriptions of external entities are often limited to meaningful names and specifications of their interfaces with the application (i.e, their input and output flows). Also, the generation and receipt of application data and/or signals by external entities may be dependent upon the application's behaviour. We use a State transition Diagrams to describe the behaviour of external entities as it pertains to their interaction with the application as in Fig. 4

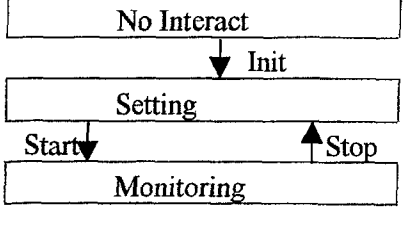

Fig. 4 State transition diagram for the External entity consumer

Specification 9: ASTS for the external entity consumer.

External Entity Transition specification is Consumer

State specification is superstate

Label specification is superlabel

Transition relation is

S $\xrightarrow{L}$ S: Superstate Superlabel Superstate

Transition axioms

1- No Interact $\xrightarrow{Init}$ Setting

2- Setting $\xrightarrow{start}$ Monitoring

3- Monitoring $\xrightarrow{stop}$ Setting

By specification 9, phase one is ended. The generating process and receiving process all of them can be done.

3.2 The Basic Ext-DFD of Monitoring state for MIES problem

3.2.1 Phase two

In this section we describe the activities carried out in phase two of our specification- generation process. In this phase, we describe how the Synchronous Interaction Specification (SIS) is generated from the Basic interpreted C-DFD [2], [14].

A. Identifying synchronous interaction

Synchronous interactions in an Ext-DFD occur across synchronous flows, during asynchronous data communication, and as a result of event occurrences. Some synchronous interactions are explicitly depicted in C-DFD's by control and synchronous flows, while others aren't. [2]

B. Specifying the Activation and Deactivation of Data Transforms.

The activation and deactivation of data transforms are specified in terms of transition on task sets. A task set consists of objects called tasks of the form <dtname, dtstate>, where dtstate is a state of the data transform named dtname. Intuitively, a task set of an application reflects the states of its active data transforms. Lables represent activation and deactivation actions are in the form:

Activation: activate (<dtname, dt>) represents the activation of the data transform named dtname with a state dt.

Deactivation: deactivate (dtname) represents the deactivation of the data transform named dtname.

Sate transforms are associated with State Transition Diagrams (STD's) which depict the effect of signals on application modes. A STD is a diagram in which states are represented by named rectangular boxes, and transition, caused by signals, are represented by directed arrows between boxes. The transitions are annotated with labels consisting of a top and bottom part separated by a horizontal line. The top part of a label gives the conditions under which the transition can take place. The conditions are expressed in terms of a single or combination of signals. The bottom part of a label states what events are communicated via the state transform's output flows; that is, what transforms are activated and deactivated, and what signals are generated to external entities and other state transforms. An example of a STD can be in Fig. 5.

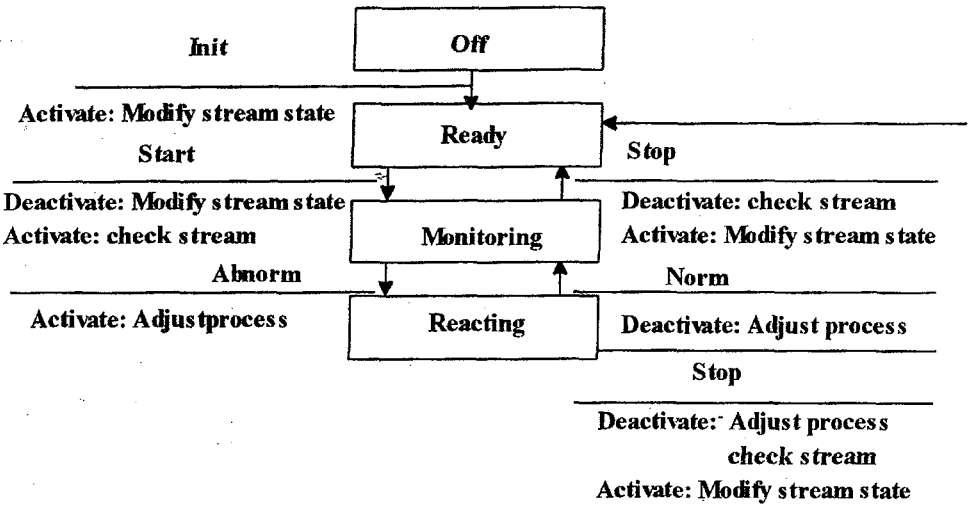

Fig. 5: State transition diagram for Monitoring process of MIES problem

C. Specifying Synchronous Interactions

An application state for the monitoring applications is of the form < dts, tp, t, ts, s, mode >, where **dts** is a task set, **tp** is a state of the data store stream state, **t** is a state of the variable stream, **ts** is a state of the queued flow Sstatus, **S** is a state of the external entity producer, and mode is a state of the transform Monmod.

Labels represent the effect of synchronous interactions on an application's state are created by the label function SYNCH, which takes a set of labels and returns a synchronous label representing the effect of synchronizing the actions represented by the labels in the set [2]. When the synchronization occurs across a data flow, a shortened form of the SYNCH label that states only the data flow name and data value will be used. **For example.**

SYNCH (Receive (flow, d), send (flow, d)) is shortened to SYNCH (flow, d).

Specification 10: A partial annotated Synchronous Interaction Specification (SIS) for the monitoring application.

Synchronous Interaction Identification is MonSynch

Synchronous Transition axioms

for all dt :dtstate ; a : taskset ; dtname : transformname ; L : tlabel

/* for data transform state dtstate two synchronization operations. First, activate operation is true according to the function is_{in} the state a in taskset returns to the state with another name and another state and has the properties of state **a**. Second, the deactivate operation which the state a be removed from the taskset */

1. **L= activate (< dtname, dt >),**

   is_{in} (dtname, a) $\Longrightarrow$ a $\xrightarrow{L}$ { < dtname, dt > } U a

2. **L= deactivate (dtname) $\Longrightarrow$ a $\xrightarrow{L}$ deletestate (dtname, a)**

   for all P , P′ : dtstate ; a : taskest ;

   dtname : transformname ; L : dtLabel

3. **P $\xrightarrow{L}$ P′ $\Longrightarrow$ { < dtname, P > } U a $\xrightarrow{L}$ { < dtname, P′ > } U a**

A1. ** Synchronous interaction between the external entity producer and the receive action of the Stream **

L1 = Receive $_{Stream}$ (Stream), S $\xrightarrow{L_1}$ S′ , L_2 = SYNCH (L1) $\Longrightarrow$

< a, tp, t, ts, Monitoring, Reacting >

< a, tp, t′, ts, Monitoring, Reacting >

B1. ** Synchronous read interaction between modify stream state and the data store stream state **

L_1 = Read (' rtstate 1' , tp), L_2 = Read ('rtstate 1', tp′)

L_3 = SYNCH (L1 , L2), a $\xrightarrow{L_1}$ a′, tp $\xrightarrow{L_2}$ tp′ , $\Longrightarrow$

< a , tp, empty V, empty q, setting, Ready >

$$\xrightarrow{L_3}$$

< a′ , tp′, empty V, empty q, setting, Ready >

B2. **Laws defining similar synchronous interactions between data stores and data transforms**

C1. ** Effect of Init event**

S $\xrightarrow{Init}$ S, L1 = activate (< 'modify stream state', (P2, null >>),

L2 = SYNCH (Init, L1), empty set $\xrightarrow{L_1}$ **a**

< empty set, tp, empty v, empty q, S, Off >

$\xrightarrow{L_2}$

< a, tp, empty v, empty q, S', Ready >

C2.....

3.3 The Behavioral specification of monitoring state for MIES problem

3.3.1 Phase three

The final phase of our specification derivation process determines the time-dependent relationships between the synchronous interactions specified in the SIS. Essentially, this phase determines what happens when application events occur in parallel. In some cases, the effects of parallel actions appear simultaneous; in others, they may be forced into a sequential order in which some event may have priority over others [13].

The **PAR** label constructor creates labels for actions whose constituent actions affect independent parts of an application, or have a well-defined net effect on the state when carried out in parallel. The following rules determine which application actions can become constituents of a parallel action [14]:

(1) Actions that affect only mutually exclusive parts of an application state - that is, independent actions can be composed by **PAR**

(2) Actions that affect common parts of an application's state can be composed by **PAR** if and only if the actions associated with the common parts have well defined net effects on the shared parts when carried out in parallel.

For example, the synchronous interaction between the receive action of the data transform checkstream and the second action of the variable stream can be carried out in parallel with the synchronous interaction between **Adjustprocess** and the **external entity** process. The send/receive interaction between the external entity producer and the variable stream can't take place in parallel with the send/receive interaction between stream and the data transform checkstream, because the effect of parallel send and receive actions on variables isn't defined -thus only sequential occurrences of access actions on variables are possible.

4- The Conclusion

In this paper a technique for associating semantics with Control-Extended DFD'S is described. The specification characterizing the semantics of a C-DFD can be viewed as formal design specification of the application modeled by the C-DFD. The formal specification can be subjected to formal manipulations facilitating their use in the rigorous validation and verification of application

behaviour. Where formal descriptions need to be created by the specifier, we have tried to make the description language as intuitively appealing as possible. The formal specification created as a result of our technique may still be unreadable by persons untrained in the specification language, but the less formal descriptions form which the specification was generated should provide intuitive notions of the semantics captured in the formal specification.

As a design tool, the technique has the potential to alleviate the problems associated with the traditional Structured Analysis/Structured Design method. A major problem cited with the use of the **SA/SD** method is the difficulty of translating **SA** specifications to **SD** specifications, and the irreversible nature of the translation. The problems can be attributed to the change in perspective as one goes from **SA** to **SD**.

The ability to use (suitable extended) DFD's as design specifications can provide a coherent, reversible transition from **SA** to design, eliminating some of the major problems associated with the **SA/SD** method. Furthermore, by associating DFD's with a less operational formal semantics suitable for use at the requirements specifications stages of software development, and a technique for verifying the operational specification against the less operational specification, one has the ability to provide a provably consistent transition from **SA** to design. A less operational semantics for DFD's has been provided by using the **LOTOS** specification language.

Appendix – A :

ASTS	Algebraic State Transition System
DFD	Data Flow Diagram
C-DFD	Control – Data Flow Diagram
EXT-DFD	Extended Data Flow Diagram
BS	Behavioral Specification
ADT	Abstract Data Types
State Spec	State Specification
Label Spec	Label Specification
SIS	Synchronous Interaction Specification
App State	Application State
App Label	Application Label
DD	Data Description
DDes	Data Domains Names
QS	Queued State
QL	Queued Label
VTS	Variable Transition Schema
DTS	Data Transform Specification
STD	State Transition Diagram
MIES	Methodology Information Exchange Service
PAR	Parallel Constructor
SA-SD	Structured Analysis – Structured Design

REFERENCES

[1] T. Demarco, Structured Analysis and System Specification. NJ: Prentice-Hall, 1978

[2] Robert. B. France, "Semantically Extended Data Flow Diagrams: A formal Specification Tool,". The institute for Advanced Computer Science, University of Maryland, IEEE no. 9107071 1992.

[3] R.B. France and T.W.G. Docker, "A formal Basis for Structured Analysis", in proc. Software Engineering' 88, 1988.

[4] D. Hatley and I. Pirbbai, Strategies for Real – Time System Specification. Dover, 1987.

[5] R.B. France, T.W. G. Docker, and C.H.E. Phillips, " Towards the Integration of Formal and Informal Techniques in Software Development Environments," in proc. New Zeland Comput. Conf. NZCS, 1987.

[6] H. Gomaa, "Software Development for Real Time Systems," commun. ACM, Vol. 29, No.7, 1986.

[7] R.B. France, "A Formal Framework for Data Flow Diagrams with Control Extensions," Ph.D. thesis, Massey Univ. (New Zealand), 1990.

[8] E. Astesiano and G.Reggio, "SMOLCS- Driven Concurrent Calculi," in proc. TAPSOFT' 87. Springer-Verlag, 1987.

[9] E. Astesiano, G. Reggio, and M. Wirsing, "Relational Specifications and Observational Semantics," in Mathematical Basis for Computer Science (Lecture Notes in Comput. Sci, Vol. 249). Springer-Verlag, 1986.

[10] T.W.G. Docker, "SAME- a Structured Analysis Tool and its Implementation in Prolog," in Logic Program, Proc. 5th Int. Conf. Symp. MIT press, 1988.

[11] E. Astesiano, A. Giovini, and G.Reggio, "Data in a Concurrent Environmnet," in proc. Int. Conf. on Concurrency. Springer-Verlag, 1988.

[12] C. Gane and T. Sarson, Structured Systems Analysis: Tools and Techniques. Prentice- Hall, 1978.

[13] T.W.G. Docker, "A Flexible Software Analysis Tool" Inform. Software Techn., Vol. 29, no. 1, 1987.

[14] R. Balzer and N. Goldman, "Prinicples of Good Software Specification and their Implications for Specification Languages," in Software Specification Techniques. Reading, MA: Addison-Wesley, 1986.

[15] G.S. Blair, L. Blair, and J.B. Stefani, " A specification Architecture for Multimedia Systems in Open Distributed Processing Systems," Distributed Multimedia Research Group, Computing Department, Lancaster University, United Kingdom. 1997.

[16] Elie Najm, Tean- Bernard Stefani, "Formal Semantics for the ODP Computational Model". Computer Networks and ISDN Systems 27 (1995). 1305-1329.

استخدام نظام مسار البيانات الممتد لفظياً كأداة مواصفات منهجية لمشكلة تبادل المعلومات (تشمل نقل الصوت والصورة)

يهتم هذا البحث بتعظيم دور الأشكال التخطيطية لمسارات البيانات واستخدامها كأداة لعمل مواصفات منهجية لتوثيق وظائف مشكلة تبادل المعلومات (تشمل نقل الصوت والصورة) وخاصة نقل الصورة فى بيئة متعددة الإمكانيات من خلال أنظمة الاتصال الموزعة بالطريقة المفتوحة عبر شبكات الحاسب وجعل هذه الوظائف مفهومة وفى نفس الوقت يمكن هذه المواصفات المنهجية عبر شبكات الحاسب وجعل هذه الوظائف مفهومة وفى نفس الوقت يمكن هذه المواصفات المنهجية من فحص الخصائص اللفظية للمشكلة وتسهيل عملية إزالة التجريد والتعقيد من المشكلة مما يؤدى إلى احتمالات نجاح للبرنامج بنسبة أكبر فى عملية التمثيل باستخدام إحدى لغات البرمجة عالية المستوى وهذا الدور البارز للأشكال التخطيطية لمسارات البيانات تم استخدامه فى تناول تطبيقات أخرى مثل قياس درجات الرطوبة والحرارة ولكن الجديد هنا استخدامه فى تناول مشكلة نقل الصورة عبر شبكات الحاسب.