Postprint of article in *ACM Software Engineering Notes* **11** (5), 53−56 (1986)

An Application of Petri Nets in Structured Analysis*

T.H. Tse and L. Pong
Department of Computer Science
The University of Hong Kong
Pokulam, Hong Kong

Background

Data flow diagrams (DFD), which originated from the popular Structured Analysis methodology [4, 5, 11] can be used to model a system in the form of a network of components and interfaces among them. When a system is too complex to be shown in a single diagram, it can be partitioned into subsystems, which can be further partitioned until each component can be described in a simple diagram made of primitive processes. Each of these successive partitions is documented in a separate DFD, so that we have a hierarchical structure. In order to ensure consistency, all data flows into/out of a child diagram must be represented on the parent diagram by the same data flows into/out of the corresponding bubble. We are also free to refine data flows into more detail on the child diagrams, so that we have a parallel decomposition of both data and processes. A labeling convention for bubbles and diagrams has been included to facilitate traceability between different levels of abstraction. Furthermore, these labels can be used for tracing the correspondence between the requirements specification and the final design.

As a tool in the requirements specification phase of a system life cycle, data flow diagrams have the following good points: It supports the creation of a modular and hierarchical structure to relax complexity. It is graphical and therefore enhances clarity. It has only a few primitives and concepts that are easily understood and used. It is behavior-oriented and provides a logical representation of the system. Parallelism is supported. Users are prevented from dealing with details too early and hence design freedom is enhanced. Requirements are expressed in terms of input and output data, so that they are testable. Data flows are paths of information flow against which performance requirements can be assigned.

In spite of its popularity, however, only a couple of automated aids [3, 6] have been developed to support the use of DFD in Structured Analysis. This is because the language is limited by the lack of a formal foundation. We must provide DFD with some formal backbone so that its usage and analysis can be computer aided.

In order to remedy the defects of informality, an attempt is made to add a mathematical structure to DFD. Petri net is found to be an appropriate model in this respect owing to the following reasons:

(a) Petri net can be represented both graphically and algebraically. The graphical representation is suitable for communications with users, whereas the algebraical representation is ideal for processing. The graphical representation of Petri nets resembles DFD in many ways. The algebraic foundation, especially the concepts of tokens and markings, provide an excellent means of analyzing the behavioral properties of information systems.

---

* This research was supported in part by a University of Hong Kong Research Grant.

(b) Petri net supports the subnet concept, so that a hierarchical representation of a system at various levels of abstraction can be created in a similar manner to that of DFD. Parallelism is also supported by Petri nets and hence irrelevant processing sequences can be suppressed, thus allowing freedom in design and implementations.

(c) According to surveys such as [2], Petri net has been found to be an excellent tool for systems design and testing, but it is not widely accepted by practitioners because it is not user-friendly. If the user-friendliness of DFD can be added to theoretical foundations of Petri nets, the resulting specification tool will have merits in both aspects.

Formal Data Flow DIagrams

Our proposed requirements specification language is known as Formal Data Flow Diagrams (FDFD). It is developed for the specification of the conceptual model of a system. The conceptual model can be regarded as the basis for other elements of a requirements specification such as performance requirements. As a result, FDFD has been designed to provide ease of reference, so that constraints and evaluation procedures can easily be associated with the appropriate parts of the conceptual model.

A system is considered as a set of elements, each of which corresponds to a user-perceived service or task of the system. These elements or tasks will be described in terms of their input, output and processing to accomplish the tasks. These tasks are related to each other through communications in the form of a network. Characteristics can be assigned to the tasks to specified performance requirements.

As discussed in [10], a requirements specification language should be graphics based and augmented by a symbolic description which is in one-to-one correspondence with the graphics. Moreover, a symbolic description is more easily input to an automated system for maintenance and analysis. FDFD has therefore been designed in two equivalent forms —– graphic and symbolic, which have corresponding syntaxes and identical semantics. The two forms can be converted from one to the other in a straightforward manner.

FDFD consists of two types of primitive elements —– data flows and tasks —– which correspond to data flows and processes, respectively, of data flow diagrams. A task in an FDFD can be decomposed into subtasks and specified in an FDFD of lower level of abstraction, so that a hierarchical specification will result. The explicit definition of input and output logic of data flows for each task is essential for an unambiguous specification. Their presence is therefore compulsory for a requirements specification in FDFD. They will be described by the "**and**" and "**or**" operators. The **and** connector of DFD fits well with Petri nets because the latter assumes an **and** operation on places connected to a transition. The **or** problem can be solved by extending the Petri net model to include input and output logic.

Let $D = \{d_1, d_2, \ldots, d_m\}$, where $m \geq 0$, be a finite set of data flows. Suppose E denotes the set of all data flow expressions over the operators **and** and **or** (such as "d_1 **and** d_2 **or** d_3"). An FDFD G is formally defined as a 4-tuple $G = (D, T, \mathbf{I}, \mathbf{O})$ such that:

(a) D is the set of data flows.

(b) $T = \{t_1, t_2, \ldots, t_n\}$, where $n \geq 0$, is a finite set of tasks.

(c) D and T are disjoint.

(d) $\mathbf{I}$: $T \rightarrow E$ and $\mathbf{O}$: $T \rightarrow E$ are functions which map tasks to data flow expressions. $\mathbf{I}$ is called the input logic function and $\mathbf{O}$ the output logic function.

To model the behavior of a system over time, we have also incorporated the notions of token and firing from Petri nets into FDFD. These dynamic elements will provide a basis for analyzing the dynamic behavior of a system directly from its requirements specification by applying Petri net

theory. Tokens can be placed in the data flows. The presence of a token means that input through a given data flow is ready for a task. A marking of an FDFD is defined as a set of tokens assigned to its data flows. It indicates the state of a system represented by the FDFD at a certain point in time. Mathematically, it is a function u: $D \rightarrow N$ from the set of data flows D of an FDFD to the set of non-negative integers N. Given an FDFD G and a marking u, we shall call the ordered couple $M = (G, u)$ a marked FDFD.

The marking can be changed by the execution of one or more tasks. A task is said to be executable if a combination of data flows satisfying its input logic functions contains at least one token each, or in other words, a combination of data satisfying the input logic is available. A marking v is said to be reachable from another marking u if there exists a sequence of executions that changes u into v.

The analysis of the dynamic properties of a system helps to detect problems which may not otherwise be apparent in the static model, such as deadlocks or tasks that will never be activated. It will also enhance the analysis of performance requirements. In addition, it can provide valuable information such as the minimum buffer space or the maximum queue length. Theory of symbolic execution may also be incorporated to simulate system behavior and provide fast prototyping for better understanding by both the user and the systems developer. However, elaborate theories on the dynamic properties of systems can be hidden from the user to avoid unnecessary complexity.

System Requirements Specification System

To demonstrate the feasibility of the language, a automated specification system based on FDFD has been implemented. It is known as the System Requirements Specification System (SRSS). It consists of a set of front end processors and an information store. The front end processors are used for the creation and validation of requirements specifications in the form of FDFD, whereas the information store is used for the storage of the resulting specifications. The front end processors of SRSS consists of the following components:

(a) The SRSS Editor is used for the creation and maintenance of system requirements specification in symbolic FDFD. It is a context-oriented editor driven by single-letter commands with or without parameters.

(b) The SRSS Syntax Analyzer performs checks on input specifications to ensure that they conform to the syntax of symbolic FDFD. It is a top-down, recursive-descent, one-symbol-plus-one-character lookahead, single-pass syntax analyzer.

(c) The SRSS Record Generator performs checking of several type of specification errors. If no error is detected, the input specification of each task is transformed into a working record in a format which can easily be accepted by the SRSS Record Storer into the information store.

(d) The SRSS Graphics Generator generates equivalent graphics representation of the input text specification.

(e) The SRSS Record Storer stores error-free and user-approved records into the information store, and performs further consistency analysis.

One important area in the analysis of a requirements specification is consistency. Consistency analysis will provide information on the completeness and correctness of a requirements specification. Following the line of [2, 1, 7] we have incorporated three types of consistency analyzes useful for requirements specifications. They are global consistency, structural consistency and behavioral consistency. Global consistency analysis helps to check whether or not the decomposition of a system into subsystems is done recursively. Structural consistency analysis checks whether any data flow entering or leaving a parent bubble is represented in a lower level diagram by the equivalent data flows into or out of some child bubbles. Behavioral consistency analysis checks whether the dynamic properties of FDFD are preserved.

A detailed specification of SRSS is given in [8], and further explanations and examples on consistency analysis can be found in [9].

Conclusion

We have applied Petri nets in the area of Structured Analysis by using it as a theoretical foundation for data flow diagrams. We have developed a specification language known as Formal Data Flow Diagrams (FDFD). The language preserves the comprehensibility of data flow diagrams and, at the same time, enables systems developers to analyze the consistency and completeness of requirements specifications.

Acknowledgements

The authors are grateful to R.K. Stamper of the London School of Economics, University of London, and S.W. Ho and C.F. Chong of The University of Hong Kong for their invaluable comments and suggestions.

References

[1] A.M. Davis, ‘‘The design of a family of application-oriented requirements languages’’, *IEEE Computer* **15** (5): 21−28 (1982).

[2] A.M. Davis and T.G. Rauscher, ‘‘Formal techniques and automatic processing to ensure correctness in requirements specifications’’, in *Proceedings of the IEEE Conference on Specifications of Reliable Software*, M.V. Zelkowitz (ed.), IEEE Computer Society, New York, NY, pp. 15−35 (1979).

[3] N.M. Delisle, D.E. Menicosy, and N.L. Kerth, ‘‘Tools for supporting structured analysis’’, in *Automated Tools for Information Systems Design*, H.-J. Schneider and A.I. Wasserman (eds.), Elsevier, Amsterdam, The Netherlands, pp. 11−20 (1982).

[4] T. DeMarco, *Structured Analysis and System Specification*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ (1979).

[5] C.P. Gane and T. Sarson, *Structured Systems Analysis: Tools and Techniques*, Prentice Hall, Englewood Cliffs, NJ (1979).

[6] G.R. Kampen, ‘‘SWIFT: A requirements specification system for software’’, in *Requirements Engineering Environments: Proceedings of the International Symposium on Current Issues of Requirements Engineering Environments*, Y. Ohno (ed.), Elsevier, Amsterdam, The Netherlands, pp. 77−84 (1982).

[7] T.J. Miller and B.J. Taylor, ‘‘A system requirements methodology’’, in *Proceedings of ELECTRO ’81*, IEEE Computer Society, New York, NY, pp. 18.5.1−18.5.5 (1981).

[8] L. Pong, *Formal Data Flow Diagrams (FDFD): A Petri-Net Based Requirements Specification Language*, MPhil Thesis, The University of Hong Kong, Pokfulam, Hong Kong (1986).

[9] T.H. Tse and L. Pong, ‘‘Towards a formal foundation for DeMarco data flow diagrams’’, *The Computer Journal* **32** (1): 1−12 (1989).

[10] T.H. Tse and L. Pong, ‘‘An examination of requirements specification languages’’, *The Computer Journal* **34** (2): 143−152 (1991).

[11] V. Weinberg, *Structured Analysis*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, NJ (1980).