DIVISION OF COMPUTER SCIENCE

A Data Flow Model for Prototyping Reactive Systems

(PhD Transfer Report)

D. A. Fensome

Technical Report No.185

April 1994

PhD Transfer Report
A Data Flow model for Prototyping Reactive Systems

D. A. Fensome

March 15, 1994

1 Introduction

To set the scene; and explain what aspect of prototyping Ïm interested in

A prototype is defined to be the first of its kind, or a model for another of its kind [McL81]. In general manufacturing, building a prototype is an accepted way of validating a design before manufacturing resources are committed for mass production. In electronic engineering prototyping is also an accepted method of design validation, often described as "breadboarding", where components are connected together to produce a working model of the system.

Applying this approach to software development has not been as easy or as well accepted in the industry as might have been expected. Perhaps this is because setting clear objectives for the prototype has not been done, or because there are many aspects of an overall system that could easily be prototyped, or even because there are a large variety of tools available for software prototyping. Sommerville [Som92] gives some examples of prototyping objectives as

- to develop user interfaces with users
- to validate system functional requirements
- to demonstrate system feasibility to management

Other objectives might be concerned with performance and user training.

More recently these objectives and the rational behind them, have been crystalised within the general description of "risk reduction" particularly by B Boehm in his seminal paper on the "Spiral Model" of software development [Boe88]. Also Ould in [Oul90] develops these ideas and describes a number of "set piece" software development process models, in particular the VP model which uses prototyping at appropriate points in the development process. Ould also points out that, as with testing, prototyping cannot prove correctness; it is a sampling technique requiring human induction to draw general conclusions.

The particular prototyping objective I am concerned with is that of functional validation, or expressed more fully the prototyping of the functional properties of reactive systems. Functional properties will describe what the system does, and may include safety and liveness properties [Lam88] and synchronisation properties, but specifically will not include performance issues related to machine execution (eg response time). The reason for excluding performance issues is that I feel these are best dealt with using target machine prototypes, or simulations of the machines, rather than a functional prototype which may run in any suitable environment. A more detailed description of the validation that may be achieved using a functional prototype may be found in [MOD90].

This is not to say that broader prototyping objectives are invalid or unimportant, but rather that I believe there are models and tools which might be particularly applicable for functional validation. Further, that there might be a model that could be useful over the whole development process from requirements capture to system maintenance, by providing a reference model for test cases.

The application domain is also important in the approach to prototyping. Harel in [Har92] describes reactive systems to include embedded, concurrent and real-time systems, but excludes data intensive systems such as databases. He says that

> reactive systems are widely considered to be particularly problematic, posing some of the greatest challenges in this field.

Thus reactive systems should be a fruitful application domain for prototyping.

Another area of confusion is the final objective of the software prototype itself; is the prototype a "throwaway" prototype (as in the traditional manufacturing sense), or is the prototype to evolve into a fully usable product? Ould describes the latter process as the evolutionary development process in [Oul90], and Crinnion in [Cri91] describes this in relation to business systems. My belief is that evolution of a prototype, in the sense of linear, direct evolution, is probably not possible in reactive systems because it is unlikely that a prototype would cover all aspects of the required system. This is especially true as reactive systems can be very large in terms of lines of code (hundreds of thousands of lines of source code being common). However this is not to say that *components* of the prototype could not be reused if this is considered to be an objective of the prototyping process.

This project is therefore concerned with prototyping reactive systems in order to check correct functionality, but with the aim of reusing some of the prototype animation components in the final production software, rather than evolution of the prototype into an operational system.

The remaining sections of this report discuss types of prototyping tools, the use of data flow as a prototyping model, my own work in animating traditional data flow models, and my proposal for a new data flow model to overcome some of the problems in prototyping from traditional data flow models. The report concludes with a proposal for further work to evaluate the new model's effectiveness in prototyping reactive systems.

2 Prototyping tools

Other methods of prototyping eg executable spec languages, Very high level languages, 4GLs and their pros and cons.

There are a wide variety of approaches to prototyping and Sommerville gives an excellent overview of existing prototyping techniques [Som92], covering many application domains. These are

Executable specification languages Techniques vary here, but include

1. executable subset of the specification language eg Objex for OBJ,
2. translation of CCS type languages into executable code, particularly finite state machines
3. translation into a logic programming language like PROLOG
4. translation into a functional programming language like MIRANDA

Diller in [Dil90] gives examples of the last two methods, and [MOD90] pp20 gives a warning about trying to execute high level specifications that are very abstract because of the danger of resolving nondeterminism in a prototype differently to a final implementation. Hayes and Jones in [HJ89] gives a similar warning, emphasising that the positive advantages of specification should not be sacrificed to the separate objectives of prototyping.

Very High Level Languages These range from functional languages to Smalltalk, APL and SETL (set based language), and also include wide spectrum languages (eg GIST and REFINE) which combine a number of standard programming paradigms. Balzer in [B+89] reports on proposals for a common prototyping language covering a very wide range of application areas, with Ada targets in mind. The proposals have an excellent overview of prototyping and its motivation.

However, none of these tools seem to have been used very much in the reactive systems area, and more generally, according to Sommerville, seem to provide problems when multiple languages are combined for prototyping purposes. Some of the executable specification problems mentioned above must also apply to using very high level languages.

4GLs Crinnion in [Cri91] and Sommerville in [Som92] describe the basis for these systems as centering on the business database, and the related query and report generation tools. Sommerville notes the lack of standards, contrasting the current state with that of real time systems which eventually lead to the development of Ada.

Composition of reusable components Sommerville illustrates this with reference to UNIX utilities, but warns that components may be too general purpose to combine effectively with other components.

Thus there are a wide variety of tools, but only some of these have been used in reactive systems, and none of them would seem to hold much prospect of component reuse in an operational system.

3 Data Flow as a Prototype Model

To explain another class of prototypes based on data flow and to briefly describe what other researchers have done.

There is another class of prototyping tool which uses data flow models as a specification basis. Very often a data flow model is used in the analysis phase of conventional development methodologies, and these models are a useful starting point for a prototype model. The dominant methods are those based on Structured Analysis [You89], but there is also JSD [Cam86]. In the UK defence industry MASCOT [Jac86] is often used, but this is more a design and implementation method based on functional decomposition, and does not have an analysis phase.

The structured analysis variants developed for real-time systems are the Ward-Mellor method [WM85] and [War86], and the Hatley and Pirbhai method [HP88]. Bowles in [Bow90] describes the evolution of structured development methods.

Coomber and Childs in [CC90] describe a protyping tool based on Ward and Mellor's data flow transformation schema [WM85], that enables syntactic checking but also assists in "proving" semantic correctness by model execution. The model uses Smalltalk as an object oriented execution environment, and uses objects communicating by messages such as "step" and "schedule", in order to emulate Ward's semantics (based on Petri nets). Coomber also proposes other execution environments such as C as an area for further investigated. The graphical features of the tool also seem important, where the approach seems to be animation of the data flow *model* rather than strictly animation of the problem domain.

Cooling [CH93] develops some ideas on the importance of "animation prototyping" in the software development process and mentions data flow specification, but seems to use SIMSCRIPT as an execution environment (presumably to provide the Petri net execution semantics of data flow in [War86]). This work also seems to concentrate on the graphical user interface, but emphasises the problem domain animation.

These researchers have used conventional discrete event simulation techniques in order to emulate data flow semantics. The disadvantage of this approach is that the prototype software is not reuseable in the final product, which could be a highly desirable objective.

Fuggetta in [F+93] takes a different approach, by defining an asynchronous communication model without an underlying control machine. The rationale here is that the execution semantics proposed by Ward in [War86] are very fuzzy, and so a data flow semantics close to data flow machine architecture is defined and used to prototype systems. The VLP specification language is used to execute data flow specifications, for business data processing applications.

Luqi has developed a language called PSDL (for prototyping system description language) [LY88] for requirement analysis, feasibility studies and the design of large embedded systems. PSDL has facilities for enforcing timing constraints and also uses "nonprocedural control constaints", as well as operator and data abstraction. Execution is possible, by translation into GIST or if the "software base contains implementations for all atomic operators and types". PSDL has evolved into SPEC [LB90] for specifying real-time systems giving a firmer base for the semantics of the event model, and Kramer in [KB93] describes the formal compositional semantics of PSDL in terms of algebraic high level Petri nets.

The JSD process network has also been used as a prototype model. Kato and Morisawa in [KM86] describe the use of a new language JSL, and a knowledge base where data types are unknown, in order to automatically generate executable code in COBOL. McNeilie in a paper in Cameron's book [Cam89] writes in a similar vein about producing executable code with a macro processor in order to define systems architecture. The work of these JSD researchers seems to be mainly concerned with automatic code generation in a business data processing environment.

Thus overall there appears to be many prototyping options even within the narrow field of data flow models, but researchers seldom seem to have prototype software reuse (in the production software) as an objective. Their objectives seem to be either the development of specification languages, the evolution of a full operational system, or the development of a simulation model as a reference for requirement capture. Also their work may be focussed on business systems.

4 My own Previous work

References to published work internal and external, including report on modelling with proprietary data flow models

My interest in prototyping reactive systems using data flow, stems from initial work with Transputers and OCCAM. It seemed to me that OCCAM was a natural tool for prototyping from data flow specifications, and also as a simulation tool. This early work is reported in [Fen90]. Developing this theme lead to work on prototyping a real-time case study, based on Hatley and Pirbhai's [HP88] Vending Machine, which is an extension of a problem Hatley and Pirbhai presented at the San Francisco Structured Development Forum in 1986. I have called the case study the Crook Proof Vending Machine (CPVM) because goods or money will not be given away to crook customers through incorrect functionality.

The CPVM also has the desired properties of the class of systems I wished to study ie event driven, with liveness properties (eg timeouts in various states), stores significant data (ie has state), and has data processing.

The work on the case study, reported in [Fen92], concentrated on the intuitively attractive approach of using Ada tasks to animate functional requirements expressed as a Hatley and Pirbhai requirement model. This approach is attractive because of the ease of mapping between a model process and an animation task, and I have called this *parallel animation*. Ada was used because of the tasking facilities, and the strong typing features which matched the data definitions required in the Hatley and Pirbhai requirement model. The work very soon showed that the overhead of emulating model data flow processes with Ada tasks

was much too large. For example the 12 CPVM functional model processes required 42 Ada tasks in the prototype, and the prototype was therefore abandoned as unmanageable. Other concurrent programming languages might have been more appropriate, for example OCCAM, but the parallel animation approach was not considered to be an effective strategy for Hatley and Pirhbai's model, even though there appeared to be an easy translation from model to prototype.

The next step was to consider *sequential animation* where some form of translation between model and animation is required, and this is reported in [Fen93b]. That report details the design and implementation of a CPVM prototype using Ada and Visual Basic as sequential animation tools. Visual Basic was used because it had good user interface facilities.

Emulation of Hatley and Pirbhai requirement model semantics with a sequential animation was certainly easier than the parallel animation, but the programming tool used for the animation did make a significant difference. Programming in an Ada environment was relatively difficult compared with the Visual Basic environment, because input polling had to be designed and some of the requirement model communication primitives were redundant (eg the model process activators). In Visual Basic however, the animation was straight forward at the design level because processing is achieved by connecting programs to input events, and the programming paradigm directly mirrors the requirement model semantics (assumed to be atomic execution by the model processes). The Visual Basic run time environment in effect carried out the event polling for the animation.

However, Visual Basic has significant disadvantages in its abstraction and encapsulation facilities. Also the benefits of the form based graphical interface are questionable for animating event oriented systems, where a tabulated list of events and consequences is probably all that is required. Other benefits of good graphics, such as ease of documentation and presentation to users, seem to me to be minimal, especially as the prototype might only be for a small part of a system (ie where there is risk or uncertainty).

I concluded that a prototype programming environment should provide

- sequential programming
- input event scanning with easy facilities for connection to the required processing
- scrolled textual I/O
- good data typing including subtypes and ennumerated types
- good abstraction facilites; at least data abstraction, but preferrably object oriented (for ease of design transfer to the final product).

During the design of these prototypes I became concerned about the semantics of Hatley and Pirbhai's data flow model in certain areas, and therefore attempted to examine data flow semantics in general, but particularly Structured Analysis (the root for Hatley and Pirbhai's method) and JSD. My finding are fully reported in [Fen93a] and summarised here.

The main difficulty with the semantics centres around the execution time for the model processes. In [WM85] Ward states

> ....the transformation executes, which should be conceived of as happening in zero time.

and then goes on to use a token based method, similar to Petri Nets, in order to understand the execution of the model. In [War86] the situation is further clouded by

> A data transformation.....is associated with an output delay, which may be zero or greater.

and Ward then goes on to consider a time step execution model with tokens, and to describe the rules for multiple token placement. This is indeed complex, and in Fuggetta's words, fuzzy!

With Hatley and Pirbhai's requirement model there is a clear statement that processes execute in zero time. However there are still some problems concerning the semantics, particularly with regard to multiple data flows. These problems can be overcome if it is assumed that when simultaneous external events occur then one of them is chosen non-deterministically and executed atomically. Within the model a segregated read/write data store must also be assumed to remove possible "race" conditions. An animation based on Structured Analysis models must reflect these semantics.

JSD models are much more difficult to reason about than Structured Analysis models, because of finite process execution time, buffered communication, and because there is some internal non-determinism in merged data streams. The difficulty arises because the effects of input events to the model may overlap during the model execution, which increases the complexity of argument about the model. In fact the process model is more an implementation model rather than a requirement specification.

JSD may also have more problems than Structured Analysis methods because of possible feedback and deadlock in the process network. However, the JSD method of separating model processes from the real world by context filtering, would seem to have some attractions in developing a physical model from a logical model (which in more direct engineering terms corresponds to "Interface Refinement"). Also using processes as objects (in the Object Orientation sense), and composing system requirements with such processes would seem to help in my objective of producing reusable software.

Another concern was whether formal techniques could be used to model the processes and state in both Structured Analysis and JSD, and also to model the external event sequences. My report [Fen93a] showed that Z was adequate for the CPVM case study, and CSP could be used to represent the external event sequence. External timing properties and liveness properties could be reasoned about by representing time as an external event into the model.

Overall however, there appeared to be a case for developing a new data flow model to aid the reasoning process, the prototype animation process, and software carry over to the systems development process.

5 The proposed 4 stage data flow model

To describe the model, and to show it overcomes shortcoming of proprietary models

As a result of my previous work it seemed to me that data flow could be the basis of a prototype specification model, and that this would be attractive because software developers are familiar with data flow and subsequent design transformations for production software. However there are difficulties with proprietary data flow models, particularly with the model semantics but also with animation transformations for a prototype. One particular area of concern was an unnecessarily complex data flow model with a supervisory control machine (the Structured Analysis models). It seemed to me that this layered approach was unnecessary, and in any case the control machine was not necessarily defined in easily identifiable external event sequences. Very often the control machine could be defined in terms of machine phases, similar to the traditional control phases of a CPU.

Another concern was the semantics of process execution, and the strategy in producing a process network (functional decomposition being the norm!). The JSD model did offer some improvements as far as the model structure was concerned, but had disadvantages with semantics and possible instability. However it seemed possible to combine the best features of both Structured Analysis and JSD in order to produce a data flow specification model which had an easy transformation to a prototype, and could provide components (spe-

cifications, designs or code) which could be reused in the production system development process. Further, that the model might be used to advantage during system development as a test oracle ie test cases could be generated and used to test production components. This has led to a proposal for a four stage data flow model which when used with appropriate animation tools, should be an effective protyping methodology for reactive systems. The essence of the model is

- to separate data processing from sequence definition. This should give a model that is easy to build and reason about, and provide components for subsequent reuse in the production development process.
- to separate physical inputs from logical inputs. This will remove an area of confusion in Structured Analysis models and segregate the essential logical inputs from implementation dependant parts.

The proposed model is shown diagramatically in Figure 1, and in more detail the four stages are as follows -

- Input Stage

  This stage provides the physical to logical conversion (or Interface Refinement) of the model. Experience has shown that such a separation provides a convenient partitioning of the model, particularly as some of the interface decisions here may be deferred to later during systems development. Also a separation of the essential logical inputs from implementation inputs will aid the reasoning process.

  Example input stage processing from the CPVM case study could be Validate_Object in Fig 2 (provides only legal coin inputs to the customer state machine); other examples could be A-D or D-A conversion, through to complex protocol conversion in distributed systems.

- Sequencer Stage

  The idea here is to provide the process stage only with events that are in their correct sequence, and to arbitrate if simultaneous input events occur. The sequencer stage is a consequence of removing the process control machine in conventional Structured Analysis models, and allows the processes in the process stage to run continuously. The definition of correct sequences will be in terms of external events, rather than in data flow machine phases (the likely consequences of the Structured Analysis models)

- Process Stage

  This stage is based on conventional Structured Analysis data flow processes with zero execution time, control and data flows and stores, but no supervisory control machine. All the processes run continuously and produce output immediately a sequenced logical input occurs. The processes could model real life objects (as in JSD), which may then be hierarchically decomposed, as with Structured Analysis, if required.

- Output Stage

  This stage provides similar functionality to the input stage viz logical to physical conversion and interface refinement. An example of this is the Money_Conversion process in Figure 2, where money values are converted to coin sequences. Again this stage could provide protocol conversion for distributed systems.

The rationale for the process semantics is based on experience in reasoning about functional requirements, particularly the desirability of removing any consideration of execution speed, data flow buffering and process activation. Also that using real life

objects as processes might lead to an object oriented design with easy component carry over to the development process. In addition it may now be possible to use parallel animation, since the tasking overhead may not now be as large as with the Hatley and Pirbhai model. Translation from the model to a parallel animation is potentially an easy method of producing an animation, since there could be a one to one mapping between model process and animation task.

The consequences of the process/sequencer machine split should be that a simpler process network results, and that testing of the separate prototype stages should be easier. However there will still be interaction between the stages particularly when errors occur, and it is proposed that any feedback should be "latched" until the next input event occurs. An example of this feedback and feedforward is shown in Figure 2 (see flows seq_error and invalid selection), which shows a functional model for the Crook Proof Vending Machine .

A useful feature of the input/output separation is that further model decomposition for distributed processing is easily achieved, once a convenient logical split has been recognised in the process stage network. The total system may also be prototyped by combining the distributed components and using the network protocol between input and output stages.

A further advantage of output separation is that additional processes for the animation can be added to this stage if required. For example visualisation of data or event sequences, rather than a simple textual representation.

Finally it may also be possible to use the model as a reference during the whole software development process, whereby developed components are validated either directly by insertion in the animation, or by application of suitable test cases derived from the model.

6 Proposed Work

To describe what is to be done and suitable methods to do it.

The work which is now proposed is to evaluate the effectiveness of the 4 stage data flow model in producing prototype animations of a new case study. The effectiveness means the ease by which existing tools may be used to produce an animation and to functionally test the prototype. A case study will be chosen that exhibits typical reactive system properties, and is different to the CPVM case study.

Particular software development activities, and their use of the model, are to be studied as follows-

- Formal Specification of requirements derived from the model and expressed in Z and CSP and animated in MIRANDA.
- Object Oriented design specifications derived from the model and animated in Eiffel.
- Target machine implementations using sequential and possibly parallel animations, using C++ or OCCAM.
- Testing Strategies eg using the animated model as a test Oracle for comparison of test results, or using assertion monitoring with the animated model and particular model components

On completion of this work refinements to the model, and perhaps alternative or modified animation languages will be suggested. Also appropriate model testing strategies will be described. Of particular concern will be the potential for prototype component reuse in the production software with consequential saving of development effort.

7 Conclusion

This report has shown the rationale for the development of a four stage data flow model for reactive systems, and proposes further work to evaluate the model's effectiveness in prototyping such systems using existing programming languages and tools. Its importance to other researchers and software developers is that a familiar data flow technique may be used to validate functional requirements, but in such a way that

- The semantics are clear
- The model can easily be translated into an animation with existing programming languages
- Components of the animation can be reused in production software
- An appropriate testing strategy may be used

The four stage model, and the evaluation of its use in prototyping reactive systems, constitute the original contribution to knowledge.

References

[B+89] R Balzer et al. Draft report for a common prototyping system. *ACM SIGPLAN*, 24(3), March 1989.

[Boe88] B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5), 1988.

[Bow90] A J Bowles. A note on the Yourdon structured method. *ACM SIGSOFT*, April 1990.

[Cam86] J R Cameron. An overview of JSD. *IEEE Trans Software Engineering*, 12(2), 1986.

[Cam89] J R Cameron. *JSP and JSD: The Jackson approach to Software development.* IEEE Computer Society Press, 1989.

[CC90] C J Coomber and R E Childs. A graphical tool for the prototyping of real-time systems. *ACM SIGSOFT*, 15(2):70–82, 1990.

[CH93] J E Cooling and T S Hughes. Animation prototyping of real-time embedded systems. *Microprocessors and Microsystems*, 17(6):315–324, 1993.

[Cri91] John Crinnion. *Evolutionary Systems Development.* Pitman, 1991.

[Dil90] Antoni Diller. *Z an introduction to Formal Methods.* John Wiley and Sons, 1990.

[F+93] A Fuggetta et al. Executable specifications with data flow diagrams. *Software Practice and Experience*, 23(6), 1993.

[Fen90] D A Fensome. The Transputer – a prototyping tool for systems. *IEE Computing and Control*, 1(1), 1990.

[Fen92] D A Fensome. Prototyping real time engineering systems using Hatley and Pirbhai's requirement model. In *Proceedings of the Xth International Workshop on Real Time Programming*. IFAC, 1992. also in University of Hertfordshire School of Information Sciences technical report No 131 April 92.

[Fen93a] D A Fensome. Modelling real time systems functional requirements using existing data flow methods. Technical Report 160, University of Hertfordshire School of Information Sciences, July 1993.

[Fen93b] D A Fensome. Sequential animation of a Structured Analysis required logical model of a vending machine controller. Technical Report 169, University of Hertfordshire School of Information Sciences, October 1993.

[Har92] D Harel. Biting the silver bullet. *IEEE Computer*, January 1992.

[HJ89] I J Hayes and C B Jones. Specifications are not necessarily executable. *IEE Software Engineering*, Nov 1989.

[HP88] D J Hatley and Pirbhai. *Strategies for Real Time System Specification.* Dorset House, 1988.

[Jac86] K Jackson. Mascot3 and Ada. *IEE Software Engineering Journal*, May 1986.

[KB93] Luqi Kramer and Valdis Berzins. Compositional semantics of a real-time prototyping language. *IEEE Trans on Software Engineering*, pages 453–477, May 1993.

[KM86] J Kato and Y Marisawa. Direct execution of a JSD specification. In *Proceedings of COMSAC*, 1986.

[Lam88] L Lamport. A simple approach to specifying concurrent systems. Technical report, DEC System Research Centre Monogram, 1988.

[LB90] Luqi and Valdis Berzins. An introduction to the specification language Spec. *IEEE Software*, pages 74–84, March 1990.

[LY88] Valdis Berzins Luqi and R T Yeh. A prototyping language for real-time software. *IEEE Trans on Software Engineering*, pages 1409–1423, October 1988.

[McL81] W T McLeod, editor. *Collins English Dictionary.* Collins, 1981.

[MOD90] Interim defence standard 00-55, part 1, requirements for the analysis of safety critical systems. MOD, 1990.

[Oul90] Martyn A. Ould. *Strategies for Software Engineering.* John Wiley and Sons, 1990.

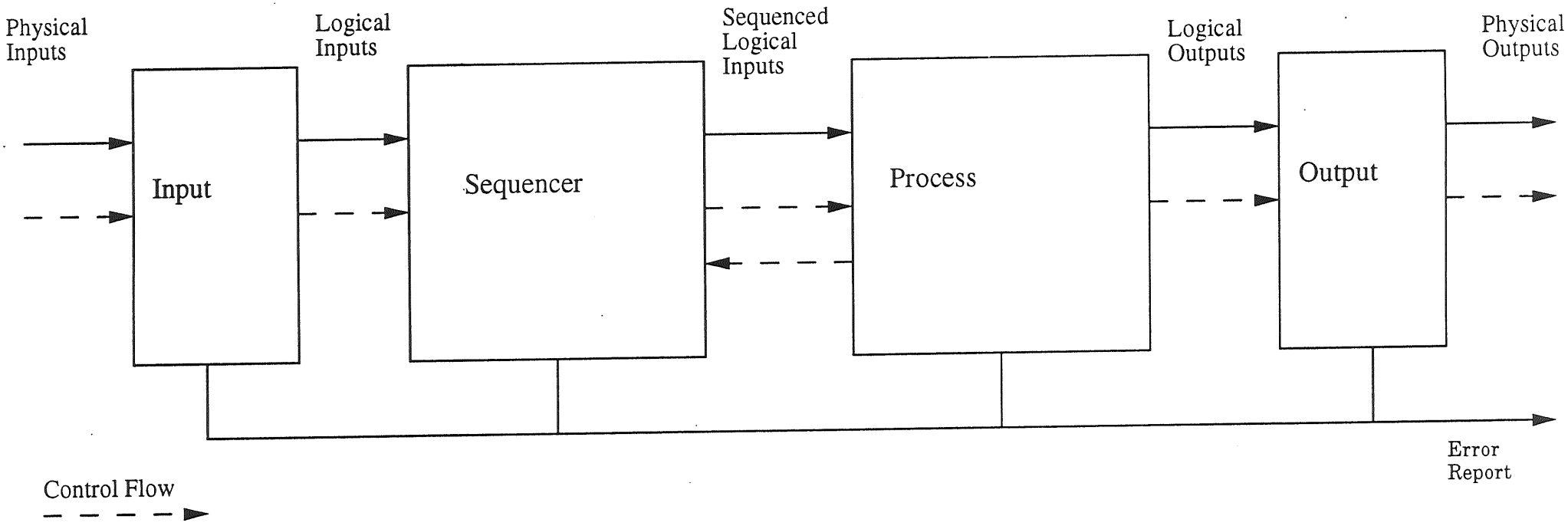
[Som92] Ian Sommerville. *Software Engineering.* Addison–Wesley, fourth edition, 1992.

[War86] P T Ward. The transformation schema: An extension of the data flow diagram to represent control and timing. *IEEE Trans on Software Engineering*, Feb 1986.

[WM85] P Ward and S Mellor. *Structured Development for Real Time Systems.* Prentice–Hall, 1985.

[You89] E Yourdon. *Modern Structured Analysis.* Prentice–Hall, 1989.

TOOLS **Z** **CSP** **Z** **Z**



4 Stage Model Structure

Fig 1

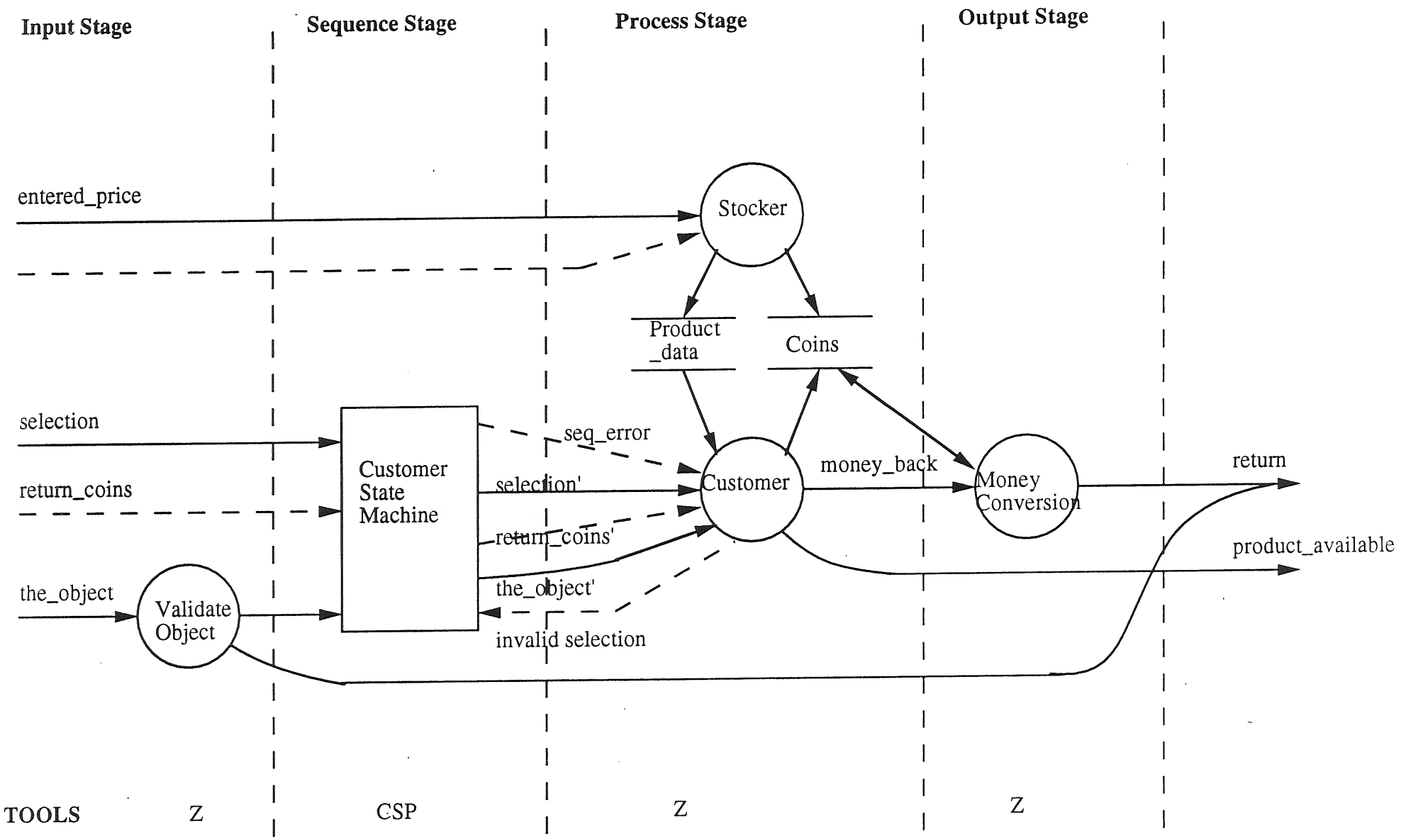


Fig 2

4 Stage Data Flow Model Level 0