QUÆSTIONES INFORMATICÆ

An official publication of the Computer Society of South Africa
and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika
en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor

Professor G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105

Editorial Advisory Board

Professor D.W. Barron
Department of Mathematics
The University
Southampton S09 5NH, UK

Professor J.M. Bishop
Department of Computer Science
University of the Witwatersrand
1 Jans Smuts Avenue
2050 WITS

Professor K. MacGregor
Department of Computer Science
University of Cape Town
Private Bag
Rondebosch, 7700

Prof H. Messerschmidt
University of the Orange Free State
Bloemfontein, 9301

Dr P.C. Pirow
Graduate School of Business Admin.
University of the Witwatersrand
P.O. Box 31170, Braamfontein, 2017

Professor S.H. von Solms
Department of Computer Science
Rand Afrikaans University
Auckland Park
Johannesburg, 2001

Professor M.H. Williams
Department of Computer Science
Herriot-Watt University, Edinburgh
Scotland

Production

Mr C.S.M. Mueller
Department of Computer Science
University of the Witwatersrand
2050 WITS

Subscriptions

Annual subscription are as follows:

<table>
<thead>
<tr>
<th></th>
<th>SA</th>
<th>US</th>
<th>UK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individuals</td>
<td>R10</td>
<td>$7</td>
<td>£5</td>
</tr>
<tr>
<td>Institutions</td>
<td>R15</td>
<td>$14</td>
<td>£10</td>
</tr>
</tbody>
</table>

Computer Society of South Africa
Box 1714 Halfway House

Questiones Informaticæ is prepared by the Computer Science Department of the
University of the Witwatersrand and printed by Printed Matter, for the Computer
Society of South Africa and the South African Institute of Computer Scientists.
ABSTRACT

A formal requirements specification language, the ELSIM language, is presented. The language uses conceptual models for modelling data processing, and control characteristics and behavior of real-time systems, and is structured into two sections, an analysis section and a design section. An extended finite-state machine model is used for modelling the control features of a system. The ELSIM language incorporates the terminology, syntax and semantics of the real-time methodology (ELSIM). The full power of the language is realised when using it within the Software Environment developed under the SEM System. The formal nature of the language allows checking for completeness and consistency of the target system specification.

1. INTRODUCTION

Real-time systems typically have a complex external interface as well as a complex inner structure. This is due to the potentially great number of interactions that may occur among the various components of an asynchronous system. Abstract models of real-time systems are generally used in order to facilitate the specification of its structure and dynamic behaviour. This is done by formally specifying how input objects determine a set of output objects. As early as 1968 a finite-state-machine (FSM) model was used to specify the requirements of the DEX-1 experimental electronic switching system [11]. Petri nets, first introduced in 1962 by Carl Adam Petri, have since been widely used to specify and verify concurrent systems [17].

The formalisation of analysis and design requirements is also generally accepted in the engineering of real-time software. Roman [19] states that significant advances in the requirements field are determined by the strength of its theoretical foundation, which is the basis for subsequent automation, and by the extent to which the theoretical results are applied in the engineering process. Teichroew [21] has performed an early survey of languages for stating the requirements of computer-based information systems, and Parnas [17] has stressed the importance of writing abstract specification of requirements which are free from implementation bias. A numbers of approaches exist to define formal specification languages [19], based mainly on experience with the design of programming languages. Work done on the development of compilers and interpreters, for which formal functional requirements are given by the syntax and semantics of the language for which they are constructed, provides a foundation for the development of requirements specification languages.

This article describes research on the use of an extended FSM model with a real-time life cycle methodology, ELSIM, and its implementation in the formal requirements and design language, the ELSIM language. The principal contributions are: the extensions to the methodology to support data modelling, and a bridge between analysis and design; and the design of a formal language based on the methodology. Section 2 addresses the idea of using FSM conceptual models for modelling real-time systems. Section 3 reviews the ELSIM methodology, Section 4 describes the ELSIM language and Section 5 presents an evaluation of the language.

2. MODELLING REAL-TIME SYSTEMS USING FSM CONCEPTUAL MODELS

Various adaptations of the basic FSM and Petri net models have been proposed in recent years as conceptual models for real-time systems. A terminal state transition diagram [15], a
graphical notation called an R-net underlying the well-known RSL language [2], and a state transition matrix for the Format and Protocol Language (FAPL) [18] are examples of this approach. In some cases extensions of the basic FSM model were used, such as the CCITT System Description Language [3] and the Requirements Language Processor (RLP) [8, 20].

2.1 Distinguishing Characteristics of Real-time Systems

Real-time systems are distinguished from other classes of systems by their degree of responsiveness. The degree of responsiveness is dependent on the criticality of the response to the operating environment requiring the response, and may range from a few seconds to fractions of a second. Real-time systems possess general characteristics, some of which are shared by all computer-based information processing systems (CBIPSSs), which are of paramount importance when modelling their characteristics and behaviour.

Essential characteristics distinguishing real-time systems from other CBIPSSs include:

• behaviour is strongly time-dependent requiring quick and correct reaction to complex sequences of external events.
• performance constraints on the system is time-dependent, i.e. maximal (no more than a specified amount of time may elapse between the occurrence of two events representing an absolute real-time constraint), minimal (no less than a specified amount of time may elapse between the occurrence of two events), and durational (an event must last for a specified amount of time).
• dynamic behaviour depends on the history of the systems and on complex logic conditions.
• output types are both data and control.
• dynamic operating rules of systems continuously change due to changes in the external behaviour of the environment, and technical advances resulting in changes in physical system configuration.
• systems are often embedded in larger systems whose primary purpose is not computation.
• physical restrictions on resources used for implementation, e.g. weight, volume, power consumption, ruggedness (resistance to changes in temperature, pressure and humidity), are often important.
• interfaces with their environment through special-purpose devices are complex, asynchronous, highly parallel and distributed.
• close coupling exists with the real world.
• testing is difficult due to time-critical dependencies.

Since real-time systems are often large and complex to understand, it is desirable that the conceptual models which are used for the development of real-time software support the principle of abstraction, and allow decomposition into its components. In order to take these characteristics and requirements into account when engineering real-time software an FSM conceptual model was considered.

2.2 An FSM Model for Real-Time Systems

The FSM is useful for conceptualising the systematic decomposition of a system into its comprehensible parts [24]. Decomposition of FSMs are usually performed according to certain criteria, e.g. to yield strongly connected, isolated sub-machines. Strongly connected machines are of both theoretical and practical interest because they have the property that any state of the machine can be reached from any other state. The analogous concept in the theory of automatic control to strongly connected machines is that of a controllable system. Decomposition of FSMs results in levels of FSMs within the system. The synthesis of FSMs to form a composite system may be done in more than one way, e.g. as a network or an hierarchy. The dynamic behaviour and functional characteristics of FSMs may be specified using state-oriented notations which express time-dependencies in a precise way. These notations are used in a number of methodologies, e.g. the SREM System [1] and the USE Methodology [23].

68
2.3 The FSM Model for ELSIM

In the ELSIM methodology, which is described in section 3, the control component of a system is called a controller. The behaviour of this controller is modelled by means of an extended Mealy type FSM [12]. The Mealy model [3] was extended to incorporate the concept of transition event, representing conditions which reflect the arrival of input signals. Enabling predicates, represented as a transition event, may be required to be true for a transition to occur. In this sense a state may be represented as a Boolean combination of conditions. A transition event may be associated with more than one state, in which case both the destination state and the transition actions may (or may not) be different. Transition actions are outputs activated when the associated transition event occurs, and are a means of specifying passage of control within the system. The set of states of a controller, the state history, represents its temporal behaviour and may be determined by tracing the state transitions among states in reaction to stimuli. The primary purpose of the FSM attributes of the system is to modify the response of the system according to past, current, and expected future conditions. The controller is distinct from the processing component of a system, which is characterised by continuous-valued inputs, outputs and internal elements.

Control is exerted on processes, or is represented by control signals to the operational environment. It does this by controlling processes (i.e. activating and deactivating them), and can be thought of in the same way as a feedback control loop in control system theory. Various state-oriented notations are used with ELSIM, all of which are formal in the sense that they are concise and unambiguous.

3. ELSIM - THE METHODOLOGY

ELSIM is the Extended Lear Siegler Inc. Methodology for engineering real-time software, which is an enhancement of LSIM, originally developed by Lear Siegler Inc. [12]. ELSIM also incorporates concepts relating to transitioning from analysis to design, based on an approach followed in the Darts Methodology [13]. It is described in detail by du Plessis [10]. In addition to the extended FSM model, the conceptual models of ELSIM include hierarchical structure, data and control flow models, functional decomposition, and a data structure model.

ELSIM tools for functional analysis are the data context diagram, the set of data flow diagrams, and process specifications for each primitive process. Data modelling is supported by the requirements dictionary and the data structure diagram. Analysis and design tools in support of real-time characteristics are reviewed below. Other design tools are structure charts, module specifications for each module, and a design dictionary.

Since the behaviour of real-time systems is strongly time-dependent and state-dependent, and the inputs and outputs of such systems are characterised by discrete-valued control signals in addition to continuous data, the modelling of the control structure, flow of control, and the passage of control within a system is of particular importance. Structured analysis tools, so appropriate for functional modelling of a system, is inadequate for representing control and temporal behaviour.

A number of control tools are used with ELSIM to this purpose. A control context diagram (ccd) is used to represent the overall control structure of a system, with components being one process depicting the system, control flows and signals and terminators. This diagram complements the data context diagram. A set of cdfs is used to model the flow of control within a process. A cfd presents the control features of processes on the corresponding dfd. A process which has an explicit controlling function, modelled as a controller, is supported by the levelled cfd and an additional tool, the control specification (c-spec). A c-spec defines the transformation of control inputs into control outputs, and provides the link between a cfd and the corresponding dfd by means of control signals called process controls. The syntax of the tools enables the flow of control between the tools to be shown. Since passage of control may depend on a complex combination of input stimuli, conditions or events, the c-spec is supported by a number of tools using state-oriented notations, namely a state transition diagram, a state transition table, a state transition matrix and a decision table.

Modules within a real-time system may execute sequentially or concurrently, requiring communication and synchronisation mechanisms. This is achieved by means of a task structuring tool called the task structure chart. A task structure chart is used to structure the
processes on the dfds into sequential and concurrent tasks, according to a number of criteria, as described by du Plessis [10]. Tasks on the task structure chart are linked by means of task interface modules, with directed edges between tasks and modules showing the data and control flows. Synchronisation between tasks, where no information transfer is involved, is achieved by means of events. An example is shown on the syntax diagram for the task structure chart in exhibit 1, to be referenced in more detail in section 4.1. The tasks on a task structure chart, each of which represents a sequential program, are structured into modules using the structure chart tool in the design part of ELSIM.

4. THE ELSIM LANGUAGE

The ELSIM language is a formal specification language for stating the analysis and design requirements of a real-time target system in a structured, unambiguous and complete manner. It is the specification language for the ELSIM Software Engineering Environment [9, 10], which was developed by means of the System Encyclopedia Manager (SEM) System of the ISDOS Project [22]. The language is based on the conceptual models of ELSIM, referred to in section 2, incorporates the semantics of the methodology and supports the analysis and design tools of the methodology explicitly.

4.1 Language Structure

The language definition was done by means of the Language Definition Manager System of SEM which requires that the language model be expressed in terms of objects, relationships and properties. This view corresponds to an entity-relationship-attribute (ERA) model [6, 7], with some difference in terminology. In this model an object type is a means of classifying data types; relationship types state how objects of the model are connected to each other; and properties are values of determinate types that describe data items for an object type. The model also allows the specification of text types which associate text strings in free format with object type instances. The data model of the ELSIM language for analysis, shown in table 1, distinguishes classes of object types according to their function in the system, namely metrics, project, data, role control and dynamics.

<table>
<thead>
<tr>
<th>MEASURES</th>
<th>PROJECT</th>
<th>ROLE</th>
<th>DATA</th>
<th>CONTROL</th>
<th>DYNAMICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>phase-product</td>
<td>controller</td>
<td>data-element</td>
<td>control-flow</td>
<td>Boolean-combination</td>
</tr>
<tr>
<td>performance-metric</td>
<td>Phase-product-part</td>
<td>engineer</td>
<td>data-flow</td>
<td>control-signal</td>
<td>condition</td>
</tr>
<tr>
<td>system-parameter</td>
<td>tool</td>
<td>process</td>
<td>data-signal</td>
<td>control-substitute</td>
<td>event</td>
</tr>
<tr>
<td></td>
<td>tool-part</td>
<td>processor</td>
<td>data-store</td>
<td></td>
<td>state</td>
</tr>
<tr>
<td></td>
<td>system</td>
<td>data-substitute</td>
<td></td>
<td></td>
<td>transition-action</td>
</tr>
<tr>
<td></td>
<td>terminator</td>
<td>group</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1
ELSIM Data Model

These classes of object types have the following meaning:

- Measures: object types enabling the specification of metrics
- Project: object types which allow instances of tools and phase products to be traced in support of project management functions
- Data: data object types within the system
- Control: control object types used
• Role: object types which perform a functional role within the system
• Dynamics: object types which enable dynamic behaviour to be specified.

The choice of object types were based on the semantics of ELSIM. The syntax and semantics of the ELSIM tools were embodied in the language, enabling the customization of the SEM tools for ELSIM. An example of the semantic model for the task structure chart is shown in exhibit 1. This syntax diagram is composed of three parts, namely the ERA diagram on the top, an instance of the task structure in the middle, and the equivalent ELSIM language statements at the bottom.

The ELSIM language is structured into two parts, namely an analysis and a design part. Each of these language parts are arranged into seven system aspects allowing various aspects of a system to be modelled. A developer is able to concentrate on each of the aspects and builds the system specification incrementally. The final system specification is a synthesis of all the system aspect specifications. The system aspects in both the analysis and design parts are: Properties and Characteristics, System Boundary and Input/Output Flow, System Structure, Data Structure, Control Structure and System Dynamics, and Data Derivation. An additional aspect for analysis is the Requirements Traceability aspect, and for design the Design Traceability and Task Communication and Synchronisation aspects. The involvement of the object types in the analysis system aspects are shown in appendix 1. Examples of the use of the language for specifying some system aspects are presented below.

The principle of abstraction is explicitly supported in both the system, data and control structure aspects. This capability allows the system, data and control components to be represented as aggregates or subparts. Abstraction is also part of the ELSIM paradigm, and is used in the analysis and design modelling tools. The various abstraction mechanisms used with ELSIM all endeavour to simplify complex representations, at first suppressing non-relevant details in the most abstract form, and gradually incorporating increasing levels of detail. Other types of abstraction are retrievable from the target system specification database, which is populated using the ELSIM language statements, by means of the Query System of the SEM System [14].

The features of the ELSIM language are demonstrated by modelling the structure and dynamics of an FSM-based controller. The example target system is an embedded ignition control module for an automobile engine. The module is named LCH ESC (Low Cost Hybrid Electronic Spark Control). The problem specification requires that the LCH ESC module be composed of a controlling part, referred to as controller asc, and a data processing part (not described here). Figure 1 shows a schematic representation of a controller asc with supporting tools. The asc-2 controller models process pl.3 in cspec-pl.3 using the ELSIM control specification tool. Control specification cspec-pl.1.3 defines the transformation of control inputs into control outputs for process pl.1.3. The control flow within process pl.1.3 is modelled in cfd-pl.1.3 using the ELSIM cfd tool. The two tool instances can be traced to each other, as is illustrated by the set of ELSIM statements for the Traceability Aspect following figure 2. The remaining part of figure 1 is interpreted in a similar manner.

Figure 2 shows the combined dfd and cfd for a process p1.3 of process asc, which is defined as cfd-p1.3 with synonym dfd-p1.3 (since these two tools have been combined here). The cfd syntax allows for the representation of control inputs and outputs entering and leaving the parent process as defined in the supporting control specification, as is evidenced by cf4 and cf3 in figure 2.
**Figure 1**
Structure of Controller asc with Associated Tools

**Figure 2**
Data and Control Flow Diagram for p1.3
A section of the requirements specification for the ignition-control module, electr-spark-control-proc, specified in the ELSIM language, is presented below by system aspect. Text descriptions of object instances are given where required to enhance readability.

/* System Structure Aspect */

DEFINE SYSTEM electr-spark-control-proc;
SYNONYM lch-esc;
DESCRIPTION;
The lch-esc is a stand-alone, micro-processor based thick film ignition-control module. Sensor input signals representing engine speed, manifold pressure and engine coolant temperature are used to calculate an optimum spark advance for a given engine operating condition. The module then uses the spark advance value to energize and fire the ignition coil primary based on an internal prediction of engine crank-shaft position. Special circuitry in the module monitors the coil primary current at spark ignition, and adaptively modifies the coil turn-on time to reduce excess current dwell and thereby reduce module power dissipation. An additional output, not related directly to ignition, has been added to the module to control manifold intake heater relay. This output is switched low, as a function of engine coolant temperature, to improve engine cold start and drive-away performance. The lch-esc module uses a custom design CMOS Angular Spark Control integrated circuit as the external timing/control interface to a standard Motorola 6805R2 microcomputer;

PARTITIONED INTO asc, sa-mp;

DEFINE PROCESSOR spark-advance-micro-proc;
SYNONYM sa-mp;
DESCRIPTION;
This processor performs the calculations of the spark advance angle, logically modelled by process p2 and its decomposed data flow diagrams. The lch-esc module uses a standard Motorola 6805R2 microcomputer to perform the calculations;

DEFINE CONTROLLER angular-spark-control-ic;
SYNONYM asc;
DESCRIPTION;
The controller is modelled on a finite-state machine and uses data from the sa-mp processor to determine control and timing data which energizes the ignition coil and fires the spark;

SUBPARTS ARE asc-1, asc-2;
MODELS p1 IN cspec-p1;

DEFINE PROCESS asc-proc;
SYNONYM p1;
DESCRIPTION;
This is the angular-spark-control process on the level 0 dfd defined as dfd-p0;

DEFINE TOOL cspec-p1;

DEFINE CONTROLLER p1.1-controller;
SYNONYM asc-1;
MODELS p1.1 IN cspec-p1.1;

DEFINE PROCESS establish-reference-time-base;
SYNONYM p1.1;

DEFINE CONTROLLER p1.3-controller;
SYNONYM asc-2;
MODELS p1.3 IN cspec-p1.3;
RELATED-TRANSFER-FUNCTION boolfunct-p1.3;

DEFINE PROCESS spark-advance-proc;
SYNONYM p1.3;
DEFINE BOOLEAN-COMBINATION boolfunct-pl.3;

DESCRIPTION;
This is the boolean function which must be true for control flow spark-advance-control to be activated;

LOGICAL EXPRESSION cf4 AND c6.c7;

DEFINE CONDITION c6.c7;
AND-PARTS ARE c6, c7;

DEFINE CONDITION engine-speed-normal-cond;
SYNONYM c6;

DEFINE CONDITION manifold-abs-pressure-normal-cond;
SYNONYM c7;

DEFINE CONTROL-FLOW angle-calculated-control;
SYNONYM cf4;

Traceability among the tools, and the components contained in a tool, are illustrated by the following statements:

/* Traceability Aspect */

DEFINE TOOL cspec-pl.1;
DESCRIPTION;
Control specification for process pl.1;

COMPONENTS ARE tdl, sttl, stdl;
TRACES TO cfd-pl.1.1;

DEFINE TOOL tdl;
DESCRIPTION;
Timing diagram representing the crankshaft position;

DEFINE TOOL sttl;
DESCRIPTION;
State transition table for the asc-1 controller;

DEFINE TOOL stdl;
DESCRIPTION;
State transition diagram for the asc-1 controller;

DEFINE TOOL cfd-pl.1.1;
DESCRIPTION;
Control flow diagram for pl.1;

DEFINE TOOL cspec-pl.3;
DESCRIPTION;
Control specification for process pl.3;

COMPONENTS ARE dt1;
TRACES TO cfd-pl.1.3;

DEFINE TOOL dt1;
DESCRIPTION;
Decision table for activating cf3;

The decision-table for the logic to obtain control flow cf3 is:

<table>
<thead>
<tr>
<th>cf4</th>
<th>c6</th>
<th>c7</th>
<th>cf3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The responsibility for a particular part of a system may be stated by:

```
DEFINE ENGINEER  acdeacon;
SYNONYM        acd;
RESPONSIBLE FOR asc-proc;
```

The traceability aspect also allows the specification of the composition of the phase products of analysis and design and traces among the tool instances which these products are composed of:

```
DEFINE PHASE-PRODUCT  lch-esc-system-req-doc;
SYNONYM        lch-esc-srd;
TRACES TO      lch-esc-dcd, lch-esc-cd;
```

and, for example, the specification of the data context diagram lch-esc-dcd:

```
DEFINE TOOL        lch-esc-dcd;
HAS COMPONENTS
p0, t1, t2, t3, t4, t5, t6, t7, t8,
i1, i2, i3, i4, i5, i6, i7,
o1, o2, o3,
cf3;
```

where t(i), i=1,2,...8 are a series of terminators in the operational environment, i(j), j=1,2,...7 a series of input signals, and o(i), i=1,2,3 a series of output signals. Control flow cf3 was defined earlier. Once again all object instances should be defined.

The following statements are the language equivalent of the composite dfd/cfd in figure 2:

```
DEFINE TOOL        cfd-p1.3;
SYNONYM          dfd-p1.3;
DESCRIPTION;
Dfd/cfd for p1.3;
COMPONENTS ARE
p1.3.1, p1.3.2,
cs2, cf1, cf3, cf4,
df1.2, df2, o2.1 df1.2, df2, df3, o2.2;
shutdown;
cs2;
```

Data flows df1.2, df2, df3 and data signals o2.1 and o2.2 are defined in the same way. The behaviour of the asc-1 controller may be illustrated using a timing diagram (td1), a state transition table (sttl), and the state transition diagram (stdl) presented in figure 3. These representations form part of the control specification cspec-p1.1, as was defined earlier.

The dynamics of the state transition from q0 to q1 are specified by the equivalent regular set of statements in the ELSIM language for the state transition table of figure 3, as follows:

```
/* Control Structure and System Dynamics Aspect */
```

```
DEFINE CONTROLLER    p1.1-controller;
SYNONYM    asc-1,
DESCRIPTION;
The definition of the finite-state machine model of this controller includes the input set I, the output set Z, the set of internal states Q, the next-state-function delta, and the output function:
I = { 0, 1 }  Z = { x0, x1 }
Q = { q0, q1, q2, q3, q4 }
delta ( q0, 0 ) = q1
   w ( q0, 0 ) = x0
delta ( q1, 0 ) = q2
   w ( q0, 1 ) = x0
```
\[
\begin{align*}
\delta(q_2, 1) &= q_3 \\
\delta(q_3, 1) &= q_4 \\
\delta(q_4, 0) &= q_l
\end{align*}
\]

\[
\begin{align*}
w(q_1, 0) &= x_0 \\
w(q_1, 1) &= x_0 \\
w(q_2, 0) &= x_0 \\
w(q_2, 1) &= x_0 \\
w(q_3, 0) &= x_0 \\
w(q_3, 1) &= x_0 \\
w(q_4, 0) &= x_1 \\
w(q_4, 1) &= x_1
\end{align*}
\]

**DEFINE STATE** q0;

**DESCRIPTION:**
This is the initial state representing the time synchronization point, which is established the instant the starter motor is activated;

**INITIAL STATE OF asc-1 WHEN c2;**
**TRANSITIONS TO q1 GIVEN tr-evtl;**
**STATE-CHANGE TO q1 ACTIVATES tr-actl;**

**DEFINE STATE** q1;

**DESCRIPTION:**
Given initial state q0, this is the state to which the controller transitions upon receiving input signal 0;

**DEFINE EVENT**
**REPRESENTS i0 AS crankshaft-0;**

**DEFINE TRANSITION-ACTION**
**tr-actl;**

**DEFINE CONTROL-SIGNAL**
**x0;**

**DEFINE CONTROL-SIGNAL**
**crankshaft-signal;**
**SYN**
**i0;**
**ATTRIBUTES**
crankshaft-pos 0;

**DEFINE CONDITION**
crankshaft-0;

**DEFINE BOOLEAN-COMBINATION**
**nextstate-funct-q1-q2;**
**NEXT-STATE-FUNCTION MAPS tr-evtl WITH q0 INTO q1;**

Another dynamics relationship, allowing the triggering of a process, is the triggers-relation:

**DEFINE CONTROL-SIGNAL**
angle-calculated-control;

**TRIGGERS** p1.3 IF angle-calc-cond;

where p1.3 is the process which calculates the spark-advance, and the predicate is defined as:

**DEFINE CONDITION**
angle-calc-cond;

Conditional transfer of control is stated by:

**DEFINE PROCESS**
p2.3.8;
**CONTROL-TRANSFER TO p2.3.5 IF c5;**

or

**EXTERNAL-CONTROL-TRANSFER TO p2.3.5 VIA tr-evtl24;**

which expresses control transfer dependence on a transition event. A complete specification will also include the definition of process p2.3.5, condition c5 and transition event tr-evtl24.

Task communication and synchronization may be illustrated by referring to the syntax diagram for the task structuring tool, in exhibit 1. Synchronization between tasks t2 and t5 is achieved by means of a synchronization module, df4-mod dependent on the arrival of event delay-ready-evtl. Task t2 is concurrent with t3. Tasks t2, t3 and t4 all access data in a data store dst2-mod. Access is coordinated by specifying that task t2 mutually excludes t3 and t4 on accessing df2 in df2-mod.
### State-oriented Notations for the asc-1 Controller

**In design**, asynchronous calls from one module to another is specified by:

#### Timing Diagram

<table>
<thead>
<tr>
<th>present state</th>
<th>input</th>
<th>actions</th>
<th>output</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>0</td>
<td>calc RPM = 1</td>
<td>(x_0)</td>
<td>(q_1)</td>
</tr>
<tr>
<td>(q_0)</td>
<td>1</td>
<td></td>
<td></td>
<td>(q_0)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>0</td>
<td>no calc = 0</td>
<td></td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>1</td>
<td></td>
<td></td>
<td>(q_1)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>0</td>
<td>no calc = 0</td>
<td></td>
<td>(q_2)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>1</td>
<td></td>
<td></td>
<td>(q_3)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>0</td>
<td>no calc = 0</td>
<td></td>
<td>(q_3)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>1</td>
<td></td>
<td></td>
<td>(q_4)</td>
</tr>
<tr>
<td>(q_4)</td>
<td>0</td>
<td>calc RPM = 1</td>
<td>(x_1)</td>
<td>(q_4)</td>
</tr>
</tbody>
</table>

**Figure 3**

State-oriented Notations for the asc-1 Controller
/* Task Communication and Synchronization Aspect */

DEFINE MODULE process-trans-mod;
SYN m1.3;
ASYNCHRONOUS CALL TO m1.3.1 WHEN c5 RECEIVING dc2;

stating that m1.3.1 is called asynchronously on receiving a control signal dc2 when condition c5
is true. Once again all object instances should be defined. Other call relationships used for the
structure chart are the d-call-relation and the d-iterative-call-relation:

DEFINE MODULE delay-hot-module;
SYN m1.3.1;
CALLS m1.3.3 WHEN delay-hot-calcd-cond RECEIVING dc1;

and

DEFINE MODULE mod1;
CALLS mod2 ITERATIVELY WITHIN mod2-labl;

with all the object instances to be defined.

The language was designed to facilitate mapping to other representation schemes, which
are close to the implementation forms for a system. A prototype of such a mapping was
demonstrated from the target database to the Ladder Diagram Programming Language, a program
design language for a programmable controller [10].

5. EVALUATION OF THE ELSIM LANGUAGE

The ELSIM language was designed to support ELSIM, a structured analysis and design
methodology for the engineering of real-time systems. The language primitives were defined to
explicitly incorporate the terminology, syntax and semantics of the ELSIM methods and tools.
The syntax of the language enforces the rigorous application of the tools when specifying the
characteristics and behaviour of a system. The language also provides support for the
transitioning from analysis to design, and offers traceability among tool products and phase
products. Since the ELSIM language was designed to be used within the automated ELSIM
Software Engineering Environment [9] under the SEM System, traceability is also supported
implicitly. The full power of the language is only realised when used within the environment,
particularly when the retrieval tools of the ELSIM SEE (and hence the SEM System) are
invoked. It has been demonstrated that the language may be used to develop specifications which
may then be mapped by automated means to implementation forms used for real-time
implementations. The issues of validation, verification and testability are under investigation at
present. The formality of the design specification provides support for testability of the
requirements and should facilitate the establishment of validation and verification criteria.

REFERENCES

1. Alford, M., [1985], SREM at the Age of Eight; the Distributed Computing Design System,
   Computer, 18,2.
2. Bell, T.E. and D.C. Bixler, [1976], A Flow-oriented requirements Statement Language,
   TRW Software series No.TRW-SS-76-02.
3. Booth, T.L., [1967], Sequential Machines and Automata Theory, John Wiley and Sons,
   Inc..
   the ACM, 30, 2.
5. CCITT, [1983], Functional Specification and Description Language (SDL),
   Recommendations CCITT Ref. Z.100-Z.104.
6. Chen, P.P., [1976], The Entity-Relationship Model - Toward a Unified View of Data,
   ACM Trans. on Database Systems, 1, 1.


**APPENDIX 1: Object types involved in Analysis System Aspects**

<table>
<thead>
<tr>
<th>Object Types</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance-metric</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>System Parameter</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Attribute</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase-product</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase-product-part</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tool</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tool-part</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data-element</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data-signal</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Data-substitute</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data-flow</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Data-store</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controller</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Terminator</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Engineer</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control-signal</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Control-substitute</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Control-flow</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Timer</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boolean-combination</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>State</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Transition-action</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Event</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Condition</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

1 Properties and Characteristics
2 Requirements Traceability
3 System Boundary
4 Input-Output Flow
5 System Structure
6 Data Structure
7 Control Structure
8 System Dynamics

**EXHIBIT 1: Syntax Diagram for Task Structured Chart**

![Syntax Diagram](image)

**Example Instance**

![Example Instance Diagram](image)

**Formal Statements**

<table>
<thead>
<tr>
<th>TASK t2</th>
<th>CONCURRENT WITH t3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MUTUALLY EXCLUDES t3, t4 ACCESS df2</td>
</tr>
<tr>
<td>TASK t5</td>
<td>SYNCHRONIZED BY df4-mod WHEN delay-ready-evt</td>
</tr>
<tr>
<td>TASK t3</td>
<td>RECEIVES-MESSAGE df6 FROM df1-mod</td>
</tr>
<tr>
<td>EVENT delay-ready-evt</td>
<td>SIGNALLED ON ARRIVAL OF df4</td>
</tr>
<tr>
<td>TASK t1</td>
<td>ACTIVATE t5</td>
</tr>
<tr>
<td></td>
<td>WRITES TO el1 df2</td>
</tr>
<tr>
<td>TASK t4</td>
<td>ACCEPT t1</td>
</tr>
<tr>
<td>TASK t4</td>
<td>PRODUCES 01, 02</td>
</tr>
<tr>
<td>TASK t2</td>
<td>SEND-MESSAGE df4 TO df4-mod</td>
</tr>
</tbody>
</table>
NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Prof. G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105
South Africa

Form of manuscript
Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. Manuscripts produced using the Apple Macintosh will be welcomed. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures
Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the author's name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Drawings etc., should be submitted and should include all relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols
Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References
References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:


Proofs
Proofs will be sent to the author to ensure that the papers have been correctly typeset and not for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters
A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.