THE SPECIFICATION OF A MENU-SELECT INTERACTIVE SYSTEM BASED ON CSP THEORY

***Arslan Enikeev*, PhD**
***Mahfoodh Bilal Ahmed Mohammed*, MSc**
Kazan Federal University, Russian Federation

Abstract

This paper presents a model for the creation of software systems using the example of a menu-select interaction system based on Communication Sequential Process (CSP) theory. This theory enables the specification and analysis of various patterns of communication between processes. The model includes the specifications of commonly used processes such as ‘stoppable’, ‘resettable’, ‘backtrackable’ and ‘coroutine’. Implementation of the model has been carried out on the basis of LISP language and it has been approved in a series of tests.

Keywords: menu-select interactive system, formal specifications, CSP theory, model

Introduction

Software systems have significantly increased in complexity and diversity in recent years, requiring the use of new, more efficient technological tools for their development. Most existing tools cannot guaranty high reliability in software products and do not allow their complete analysis. We propose a new model for the study of the properties of these products, starting from the initial description of the problems they are programmed to solve and also examining all stages of their development.

These problems can be solved using formal mathematical models which provide a rigorous approach to software development. However, the experience of software development shows that the use of formal methods in the design of software systems often leads to cumbersome constructions which cause a serious obstacle to the development process. It follows that we need to create an appropriate conceptual apparatus which will make these formal methods more applicable to software development in practice. Considering this problem, of the various tools for creating models available, the most appropriate one seems to be the theory of communicating sequential processes, or CSP (Hoare, 1985), which, by using the conceptualization of sequential processes, enables the specification and analysis of various patterns of communication between processes (including parallelism). The principles of process specification and analysis in CSP are consistent with the top – down method of development, and therefore permit reductions in the complexity of formal methods of software development by abstracting all non-essential parts of the process.

An advantage of using CSP facilities is that they allow the combination of formal methods with the programmer’s intuition to create software tools for the computer assisted development of software application. In particular, C.B. Jones, in his monograph (Jones, 1980) proposed in cases of high complexity to replace the formal proof of program correctness by so-called correctness arguments, which can be formulated based on the experience and intuition of the programmer. One of the important stages of a rigorous approach to software development is the construction of a formal model of the application being created based on a suitable mathematical apparatus. In this paper we present the specification of a menu - select interactive system model using CSP.

Menu-select interaction

By an interactive system we mean a programming system in which interaction between the system and user is realized in the form of a question and answer. The main purpose of interactive systems is an optimization of human-machine interaction when problem solution cannot be completely formalized before execution, or the methods of their solution can be substantially improved by dialogue interconnection. The distinctive feature of this sort of problem is that the subsequent stages of their solution can be changed depending on the results of previous stages and usually cannot be chosen automatically by system programs. The most typical examples of systems intended to provide the above mentioned facilities, are managing, teaching, automatic design, information search and problem solving systems.

The first objective of dialogue interconnection is to prompt, invite and guide the end user to input all the data correctly. End users will particularly need interactive system guidance if they are occasional users, or if the data structure is complex. There are a number of different methods of providing this guidance of which the most appropriate is menus. By menu we mean lists of options from which the user may select by moving a cursor or entering the appropriate keyword. An interactive system based on the menu select interaction principle is called a menu-select interactive system. The most essential benefits provided by a menu select interactive system are the following:

1. Menus help the user to clear the hurdle of getting started.
2. Users do not have to type a syntactically structured request. Instead, they only select from a list of options.
3. The user can go back up to any of the menus used on the previous steps to change the progress of problem solution.

The behavior of a menu–select interactive system can be modeled as a set of sequences of possible responses. This allows description of the menu-select interactive system model using CSP theory. In CSP notation these sequences are called traces. A **trace** is a finite sequence of symbols recording the actual or potential behavior of a process from its beginning up to some moment of time. Each **symbol** denotes a class of events in which a process can participate.

The set of symbols denoting events in which a process can participate defines the **alphabet** of a process. A process is defined by the set of all traces of its possible behavior. From the definition of a trace, it follows that process P with alphabet A;

P0. $P \subseteq A^*$, where A^* denotes the set of all traces with symbols from a pre-defined alphabet A;

P1. $<> \in P$, where <> denotes an empty trace;

P2. $st \in P \Rightarrow s \in P$, for all $st \in A^*$,where st is the concatenation of s with t

Below we present some important definitions from CSP that will be used subsequently.

If s is a nonempty trace, we define s_0 as its first symbol, and s’ as the result of removing the first symbol from s.

Let √ be a symbol denoting successful termination of the process. As a result, this symbol can appear only at the end of a trace. Let t be a trace recording a sequence of events which start when s has been successfully terminated. The composition of s and t is denoted (s; t). If √ does not occur in s, then t cannot start.

If s is a copy of an initial subsequence of t, it is possible to find some extension u of s such that su = t . We therefore define an ordering relation $s \le t =_{df} \exists u\ (su=t)$ and say that s is a *prefix* of t . For example, $<x,y> \le <x, y, z>$, $<> \; < \; <x, y>$.

The ≤ relation is a partial ordering, and its smallest element is <>.

The expression $(t \lceil A)$ denotes the trace t when restricted to symbols in the set A; it is formed from t simply by omitting all symbols outside A. For example,
$<a, d , c, d> \lceil \{a,c\} =<a, c>$.

The expression P^0 denotes the set of first symbols of all traces in process P (initial state of process P). To put it formally: $P^0 =_{df} \{ c \mid < c > \in P \}$.

Process FAIL $=_{df}$ {<>} , which does nothing, process SKIP $=_{df}$ {<>,<√ >}, which also does nothing, but unlike FAIL it always terminates successfully.

Let x be an event and let P be a process. Then (c → P) (called ‘c then P’) describes an object which first engages in the event c and then behaves exactly as described by P. The process (c → P) is defined to have the same alphabet as P; more formally,
(c → P) $=_{df}$ {c→ s | c ∈ αP & s ∈ P }, where c→ s $=_{df}$ < c > s, αP denotes an alphabet of process P.

Let P be a process and s ∈ P then P / s (P after s) is a process which behaves the same as P behaves from the time after *P* has engaged in all the actions recorded in the trace *s*.
If s ∉ P, P / s is not defined; more formally, P / s $=_{df}$ {t | st ∈ P}.

Let P and Q be processes. The operation P | Q is defined as following:
P | Q $=_{df}$ P ∪ Q , where α (P | Q) = αP ∪ αQ (the choice between P and Q).
The choice depends on which event from $(P \mid Q)^0$ occurs. For example, if R=(a → P) | (b → Q), R/<a> =P and R/< b > = Q

Let P and Q be processes. Sequential composition P; Q is defined as a process which first behaves like P; but when P terminates successfully, (P; Q) continues by behaving as Q. If P never terminates successfully, neither does (P; Q). More formally, P;Q $=_{df}$ { s;t | s ∈ P & t ∈ Q }.

In CSP a menu select interaction can be specified as communicating process P. The initial menu, with a set of events, is displayed on the screen, represented as P^0. After the user has selected one of these events, say x ($x \in P^0$), the subsequent interaction is defined by
P/< x.> (P after x), i.e. $(P/< x.>)^0$ … .

A set of menus and interactive prompt representations are provided for a set of functions logically used together. Each symbol in the menu denotes a function, invoked after the user’s selection. We will make a distinction between the commonly used functions, controlling the interaction process, and the problem dependent functions, the choice of which can be defined depending on the particular sort of problems.

The most typical commonly used functions of the menu-select interaction are the following:
functions for terminating or quitting a process;
functions allowing the return to any of the previous steps;
switch-functions – for switching to another process, in particular, to use another set of menus or command and data entry, for which the menu technique is less suitable.
The main objective of this paper is the specification of an abstract menu-select
interaction on the basis of CSP, concentrating our attention on these commonly used functions.
This paper investigates the following commonly used functions:
‘stop’ – to terminate or quit a process;
2.1. ‘reset’ – to start again from the beginning of a process;
2.2. ‘back’ - to undo the most recent action in a process;
3.1. ‘off’ – to pause a process and switch to another;
3.2. ‘on’ – to restore the action of the most recent process interrupted by the ‘off’ -function.

The model of a menu-select interactive system is based on the specification of these functions, which can be described in CSP. But CSP facilities are not enough to describe a menu-select interaction model completely. Therefore we need to extend CSP facilities with new processes which define the above mentioned functions.

If ‘P’ is a process, let’s define the following processes:
2.1. Stoppable (P).

Let ‘stop’ be a symbol not in the alphabet αP. Then a process stoppable (P) can be defined as a process which behaves like P, except that

‘stop’ is in its alphabet;
‘stop’ is in every menu of stoppable (P);
when ‘stop’ occurs, stoppable (P) terminates successfully

For example:
<a, b, c, stop, √ > ∈ stoppable (P) ⇔ <a, b, c> ∈ P, where symbol √ denotes the event of a successful termination of the process.

2.2.Resettable (P).

Let ‘reset’ be a symbol not in the alphabet αP. Define resettable (P) as a process that behaves
like P, except that
‘reset’ in its alphabet;
‘reset ‘ is in every menu of resettable (P);
when ‘reset’ occurs, resettable (P) starts again from the beginning

For example:
<a, b, reset, c, d>∈ resettable(P) ⇔ <a, b>∈ P & <c, d> ∈ P

2.3. Backtrackable (P).

Let ‘back’ be a symbol not in the alphabet αP. Define backtrackable (P) as a process that behaves like P, except that
‘back’ in its alphabet;
back is in every menu of backtrackable (P);
backtrackable (P)/s<x,back> = backtrackable (P)/s provided x ≠ ‘back’
The intention is that ‘back’ will cancel the effect of the most recent action which has not already been cancelled (other than ‘back’ itself).

For example:
<a, b, back, d>∈ backtrackable (P) ⇔ < a, d >∈ P
<a, b, c, back, back, d>∈ backtrackable (P) ⇔ < a, d >∈ P

2.4, Coroutine (P).

Let $(\alpha P \cup \alpha Q) \cap \{off,on\} = \varnothing$. Define coroutine (P,Q) as the following:
initially coroutine (P,Q) starts from the process P and then
when ‘off’ occurs it is switched to the process Q which starts from the beginning, and then
when ‘on’ occurs it is switched from the process Q to the process P, continuing from the point, at which process P had been interrupted;
the subsequent behavior of ‘coroutine (P,Q)’ is defined as following:
(2.1) when ‘off’ occurs it is switched from process P to process Q, continuing from the point at which process Q had been interrupted;
the same as in (1.2).

The behavior of ‘coroutine (P,Q)’ can be illustrated as following:

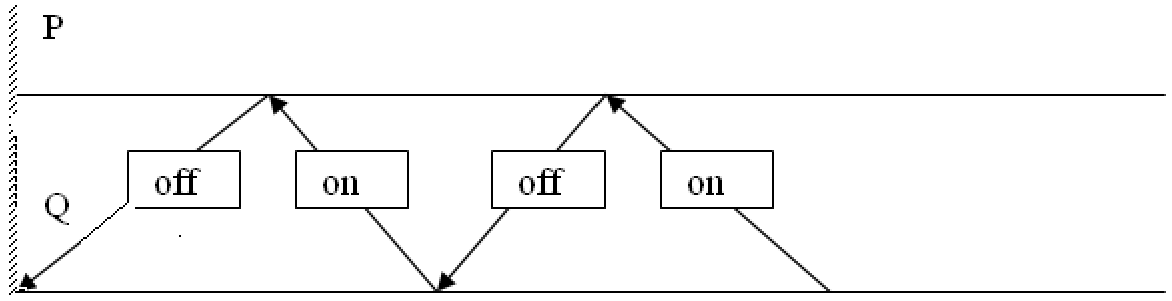

Figure 1.

The coroutine (P,Q) process is an important one, because it permits the description of such well-known processes as lexicographic analysis and parsing in compilers, multiprogramming, time sharing and other important processes.

The specifications and properties of commonly used operations

Interactive systems are more difficult to specify and analyze than other concurrent systems. This is because they require the conceptualization not only of interactions between their subsystems, but also of the complex interactions between the user and the system. It is clear that the programmer’s intuition is not enough, being unreliable in cases of high complexity. One of the important problems in system specification and analysis is the definition of process properties and their proofs using a satisfactory conceptual apparatus. This chapter concerns the specifications and proofs of the properties of the main commonly used processes of menu-select interactive systems using CSP. The ‘stoppable’, ‘resettable’, ‘backtrackable’ and ‘coroutine’ process operations are considered. These operations introduce new commonly used actions to supplement arbitrary menu-select applications. We stipulate that all processes controlled by the above mentioned commonly used operations are well terminating in the sense that they do not progress towards FAIL process. FAIL process represents a state after the run time errors such as divergence, deadlock and non-termination. A more formal definition is:

P is WTP (Well Terminating Process) $\Leftrightarrow$ $\forall s \in P. \ (P/s <> FAIL)$
This assumption is important for the implementation of operations.

In this paper we intend to use the following styles of process definition:
a trace based definition, defining a process as the set of its traces;
a derivative definition, defining a process P by two objects:
a set I, defining the initial state of process P , i.e. P^0;
a function mapping each member ‘c’ of I into a process, defining the subsequent behavior of P, i.e. P/<c>

To prove the relevant properties it is reasonable to use the first definition style. Derivative definitions are useful for the implementation of processes, because they directly define their behavior. Below we formally define the commonly used processes based on the above mentioned styles of process definition.

3.1. Definition of ‘stoppable’.

Definition 1 (trace based).

$\text{stoppable}(P) =_{df} \{st \mid s \in P \ \& \ t \in (stop \to SKIP)\}$), where P is WTP & $stop \notin \alpha P$

Definition 2 (derivative).

$stop \notin \alpha P \&$

$(\text{stoppable}(P))^0 = P^0 \cup \{stop\}$ &

$$(\text{stoppable}(P))/\langle x\rangle = \left|\begin{array}{l} SKIP, \text{ if } x=stop \\ \text{stoppable}(P/x), \text{ if } x \neq stop \end{array}\right.$$

The last definition can be adequately implemented in the form of a recursively defined function or procedure.

3.2. Definition of ‘resettable’.

Definition 1 (trace based).

$\text{resettable}(P) =_{df} \{s \mid s \in (\alpha P \cup \{reset\})^* \ \& \ (\forall s_1 <= s) \ \text{rest}(s_1) \in P\}$, where P is WTP& $reset \notin \alpha P$, $\text{rest}(s)=s$, if $s \upharpoonright \{reset\} = <>$,

$\text{rest}(s)=s_2$, if $s= s_1 <reset> s_2 \ \& \ s_1 \upharpoonright \{reset\} = <> \& \text{last}(s_1) \neq \surd$

$\text{last}(u<a>)=a$

Definition 2 (derivative).

$reset \notin \alpha P \&$

resettable(P) =start(P,P), where

$(\text{start}(P,Q))^0 = P^0 \cup \{reset\}$ &

$$(\text{start}(P,Q))/\langle x\rangle = \left|\begin{array}{l} \text{start}(Q, Q), \text{ if } x=reset \\ \text{start}(P/x, Q), \text{ if } x \neq reset \end{array}\right.$$

The last definition can be adequately implemented in the form of a recursively defined function or procedure.

Definition of 'backtrackable'

Definition 1 (trace based).

backtrackable(P) $=_{df}$ $\{s \mid s \in (\alpha P \cup \{back\})^* \,\&\, (\forall\, s_1 <= s)\ clean(s_1) \in P\}$, where

P is WTP & back $\notin \alpha P$,

clean(s) =s, if $s \upharpoonright \{back\} = \langle\rangle$,

clean(s) = clean(s_1 s_2) if s= (s_1<a, back> s_2) & a $\notin$ {back, √}.

clean(s) = clean(s_1) if s= < back> s_1

Definition 2 (derivative).

back$\notin \alpha$P&

backtrackable(P) =recover(P,P), where

$(recover(P,Q))^0 = P^0 \cup \{back\}$ &

$$(recover(P,Q))/\langle x\rangle = \begin{cases} Q, & \text{if } x=back \\ recover(P/x, recover(P,Q)), & \text{if } x \neq back \end{cases}$$

The last definition can be adequately implemented in the form of a recursively defined function or procedure.

Definition of 'coroutine'

The trace based definition of 'coroutine' seems to be too complicated. So here we will only consider a derivative style of definition:

(P is WTP) & (Q is WTP) & $(\alpha P \cup \alpha Q) \cap \{off, on\} = \varnothing$ &

$(coroutine\ (P,Q))^0 = P^0 \cup \{off\}$ &

$$coroutine\ (P,Q) = \begin{cases} coroutine1\ (Q,P), & \text{if } x=off \\ coroutine\ (P/\langle x\rangle,Q), & \text{if } x \neq off, \end{cases}$$

where

$(coroutine1(Q, P))^0 = Q^0 \cup \{on\}$ &

$$coroutine1(Q, P) = \begin{cases} coroutine\ (P,Q), & \text{if } x= on \\ coroutine\ 1\ (Q/\langle x\rangle,P), & \text{if } x \neq on \end{cases}$$

The properties of commonly used operations

A study of process properties is one of the main problems of a formal approach to software development. A study of the respective properties of the task to be performed enables the selection of the most reasonable and efficient development path. Formulation and proof of these properties helps to avoid making unfavorable decisions, which is very important especially in the initial stages of software development. Study of properties is based on a mathematical model which allows secure, unambiguous, precise and stable process specification. Here we present an example of property analysis of the commonly used 'resettable' operation using CSP facilities. We will consider the following properties:

(1) if P is a process then resettable(P) is a process;

(2) $s \in P \,\&\, s \upharpoonright \{\surd\} = \langle\rangle \Rightarrow (resettable(P)/s)^0 = (P/s)^0 \cup \{reset\}$

(3) s <reset>$\in$ resettable(P) => resettable(P)/s <reset> = resettable(P);

(4) $s \in P \,\&\, s \upharpoonright \{\surd\} = \langle\rangle$ => resettable(P)/s= start(P/s, P);

(5) s$\in$ resettable(P) => resettable(P)/s= resettable(P)/rest(s)

We omit the proofs of the above mentioned properties. Similar properties could be presented for other commonly used operations.

Conclusion

This paper presents an approach to the creation of mathematical models for software systems, giving the example of a menu - select interaction using CSP facilities. The model can be used for specification, proof of process properties and implementation. The implementation of the model was carried out on the basis of LISP language and has been approved in a series of applications. We do not consider these implementations here as we hope to present them in the next paper. CSP theory permits the creation of models for a variety of software systems and is especially appropriate for those that are based on an event-driven programming paradigm. This paradigm is widely used in the majority of object–oriented programming systems. The evolution of the object-oriented programming technique has caused the appearance of the new CSP-OZ theory (Fischer, 1997, Enikeev, Benduma.2011), which is based on a combination CSP and object-oriented specification language Object-Z (Duke, 1995). This new theory provides a specification of the behavior of communicating processes and in addition to CSP permits the description of object-oriented models.

References:

C. A. R. Hoare, Communicating sequential processes, Prentice-Hall, Inc., Upper Saddle River, NJ, 1985

J o n e s C. B. Software Development. A Rigorous Approach, — Prentice Hall International, 1980. 382 p.

C. Fischer. CSP-OZ: A combination of Object-Z and CSP. In H. Bowman and J. Derrick, editors, Formal Methods for Open Object-Based Distributed Systems, volume 2, pages 423–438. Chapman & Hall, 1997.

Арслан Еникеев, Тахар Бендума, Специализированные модели для разработки информационных систем, изд-во LAP, LAMBERT Academic Publishing, ISBN: 978-3-8454-4045-3, 2011.

R. Duke, G. Rose, and G. Smith. Object-Z: A specification language advocated for the description of standards. Computer Standards and Interfaces, 17:511–533, 1995