Journal of the University of Chemical Technology and Metallurgy, 47, 5, 2012, 565-569

AN ALGORITHM FOR SYNTHESIS OF PROCESS-ORIENTED SOFTWARE AND ITS DISTRIBUTION ON SINGLE AND MULTIPROCESSOR SYSTEMS

A. Atanassov

University of Chemical Technology and Metallurgy
8 Kliment Ohridsky, 1756 Sofia, Bulgaria
E-mail: naso@uctm.edu

Received 27 July 2012
Accepted 12 October 2012

ABSTRACT

The algorithm given in the paper is based on the theory of Communicating Sequential Processes /CSP/ and on the experience of the author in the development of real-time software systems and applications. The CSP theory presents the software or hardware systems as a number of communicating processes, exchanging messages or events via channels. Right grouping and mapping of the CSP processes to the software processes and threads, and their allocation on the microprocessors will dramatically increase the performance of the whole system. The paper presents a CSP process-oriented approach to system decomposition and main algorithm steps for processes' and threads grouping or splitting in order to meet the time constraints defined for the system.

Keywords: CSP theory, process oriented software, operating systems processes & threads.

INTRODUCTION

The theory of Communicating Sequential Processes /CSP/, developed by A.S. Hoare [1] is a mathematical formalism - the process algebra describing the behavior and interactions between components of a wide range of systems. The basis of the theory are the processes exchanging sequences of messages (events) via input and output channels.

The process is a component, encapsulating some data structures and algorithms that are inaccessible to other processes. They are private in the terms of object-oriented programming /OOP /. Each process executes its own algorithm in one or more threads, scheduled by the operating system /OS/. In control systems the process can be related to the controlled object, to the controller or to various filters, adders, nonlinear elements, etc.

From a more abstract point of view the process can be regarded as the finite machine, driven by events (or data) received from other processes via input channels.

Each process has an alphabet (interface), described by many events in which the process may be involved or is able to interpret. If the process P can interpret events a, b, c and d, then the alphabet αP is written as:

$\alpha P = \{a,b,c,d\}$

At a certain point of time the process is able to interpret some of the events of the alphabet, or to reject the interpretation of some of them. An event represents an action that can be enforced by a process (a specific step of the algorithm or process change from one state to another).

An event can be combined with a process using the prefix operator (->). Process bang -> UNIVERSE first interprets the event bang and then behaves like UNIVERSE. This new process can be given the name CREATION.

CREATION= bang -> UNIVERSE

Alternative participation of a process in several sequences of events can be described

as: *P = a -> b -> c -> STOP | d -> STOP,*

or P= *P(a) -> P(b) -> P(c) ->STOP | P(d) ->STOP,*

where the sign | is an operator to choose between sequences, a P (x) e R participation process in the event x.

In compositional relation the processes can be combined:

· Sequentially (operation - ;)

CAR= Start ; Drive ; Finish

· In parallel (operation ||)

$$PAR(V) = \Big\|_{i=1}^{n} (P_i, A_i)$$

· In parallel with priority (operation $\overset{<-}{\|}$).

The latter is analogous to parallel as described above, but the process standing on the left of operator is implemented with a higher priority than standing on the right. This statement makes sense in a real-time computer system where the processes with higher priority are implemented as an OS process or a thread running with higher priority.

Processes communicate via channels through which it must be possible to structure events. Channels are characterized by name, data type and direction. Channels are always unidirectional (point-to-point) and are either input or output from one process to another.

An output channel of the process (*P*) can be presented as:

c ! x -> P, x∈ T

An input cannel of the process (*Q*)) can be presented as:

c ? x:T -> Q(x),

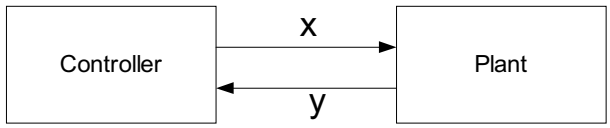
where *c is* the name of the channel, *T* is the channel type, and *x* is a variable (event) from type *T.* The sign ! indicates the output of data, with the sign? - data entry.

Very often when describing the CSP processes the CSP and charts are used, as given in the example below, or process diagrams of the type:

Process(in, out) = in -> out -> Process(in, out),

where *in, out* can be channels or a subset of events related to the process or data of a certain programming language.

The following example provided by Hilderink [2] illustrates the application of the theory of CSP in the design of controllers for a given unit. Fig. 1 below presents a summary chart of a CSP-control system (System), consisting of a working in parallel controller and an unit (Controller and Plant):



$$System = Controller \,|_{\{x,y\}}|\, Plant$$

Fig. 1. CSP-chart of a system including a controller and a unit (plant).

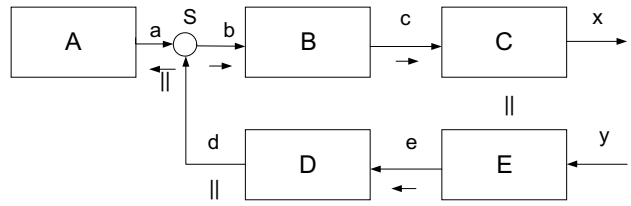


Fig. 2. CSP chart of the Controller.

The structure of the controller is given on Fig. 2.

The processes and the channels describing the controller can be presented compositionally as follows:

A(a) *S(a,b,d)* *B(b,c)*

C(c,x) *E(y,e)* *D(e,d)*

Given the requirement that the process A must have the highest priority and be implemented in parallel with the sequential processes S, B and C, and in parallel with the sequential processes D and E, the final recording of the regulator takes the following form:

$$Controller = A(a) \overset{<-}{\|} ((S(a,b,d) ; B(b,c) ; C(c, x)) \| (E(y,e) ; D(e,d)),$$

where for example process *B* can be described as:

$$B = b?\, z \rightarrow c\, !\, f(z) \rightarrow B.$$

Through the process algebra, the CSP complex systems which surround us, can be described and presented as compound-component processes that interact concurrently or are constructed from other sequential or parallel processes, etc. Using the process oriented approach, complex processes can be decomposed to sub-processes and their input-output channels to be specified together with the messages (events) that they communicate. A CSP-description of a system contributes to its detailed design, programming and verification of its performance

The refined processes then have to be mapped to operating system processes and threads.

The last ones are not subjected of the CSP theory. They are primitives of the OS. The process or the application or the task are synonyms and are related to an executable unit (file) which can be managed or scheduled by the OS.

OS Process. The process has priority, ID, its own stack, data block, handles, etc., and minimum one thread, which executes the code of the process. The process runs in its own memory space.

OS Threads. The threads are existing inside the process. They also have ID, priority and other attributes but not their own memory. They share the process memory.

Both threads and processes can be scheduled by the OS. Switching from one process to another (known as context switch) consumes more processor time than switching between the threads inside the process. Different OS use different scheduling algorithms (schedulers): round-robin, fixed priority, preemptive, non-preemptive, etc. When CSP processes are mapped to OS threads and processes, it is very important how to choose their priority. Assigning wrong priority to a thread or process will impact the entire system performance. This is valid also regarding the choice of the OS (Windows, Linux, Open Embedded, QNX, etc.). Some of them are not appropriate for real-time systems and can not guarantee the sampling required from hard real-time applications.

DESCRIPTION OF THE ALGORITHM AND ITS STEPS

1. Determination of the overall control process or overall software system.
2. Decomposition of this process to sub-processes on the basis of the functional requirements and constraints (time, hardware, software, etc.). In this case interdisciplinary knowledge in control systems, parallel programming, OS and real-time OS is required.
3. Definition and determination of the channels and the messages exchanged over these channels (alphabet of the processes).
4. Determination of processes' priorities (from the control point of view)
5. Refinement of algorithms of individual processes.
6. Selection of a software programming architecture and mapping the CSP-processes to OS processes and threads.
7. Distribution of the OS processes over hardware (processors and/or cores) .

For process allocation the following formula can be applied:

$$Ttotal = Tos + \sum_{i=1}^{N} (Tproc_i + Tswitch_i) < Td$$

where:

$Ttotal$ – is a total software execution time;

Tos – is time for OS;

$Tproc_i$ – is the execution time of the process i ;

$Tswitch_i$ – context switch time for process i;

Td – is sampling time of the control system or critical time of the software system.

RULES FOR MAPPING AND ALLOCATION OF THE CSP PROCESSES TO OS THREADS AND PROCESSES, AND ON THE PROCESSORS:

- IF $Ttotal < Td$, THEN the processes can be executed to one processor or core.
- IF $Ttotal > Td$, the processes can be executed to N processors or cores, or to a faster processor $N = Ttotal/Td$ (N is the bigger integer number)
- IF processes $Proc_i$ and $Proc_j$ are working with shared data THEN they can be formed as threads in one OS process.
- IF processes $Proc_i$ and $Proc_j$ are not working with shared data THEN they can be formed as separate OS processes.

The execution time of a certain process $Tproc_i$ includes communication times $Tcomm_{ij}$ to other processes and threads, where j varies from 0 to N-1.

In order to determine the above mentioned times two programming classes - Statistics and Logger, were developed. They serve to record the statistics associated with threads and processes of the developed software systems. The statistics class register the IDs of OS threads and processes, their priorities, their names (defined by the programmer), their time for communication with external processes, their time to work on common data and more. The class Logger records statistical information in log files, allowing recording of events / information with different priority (up to 8 different levels of importance). Off-line analysis of the log files information is used in the decision to map a process on a thread or how to allocate the processes on the processors or cores.

EXAMPLE OF THE ALGORITHM USAGE

The next example demonstrates how the algorithm can be applied to form process-oriented architecture of one WEB-based system intended to maintain and diagnose of letters sorting systems in USPS [3]. The system is known as Local Performance Diagnostics Server /LPDS/. A simple view of the system is given on Fig. 3.

It consists of an WEB server, an WEB component (WebUI), some diagnostic components for SW and

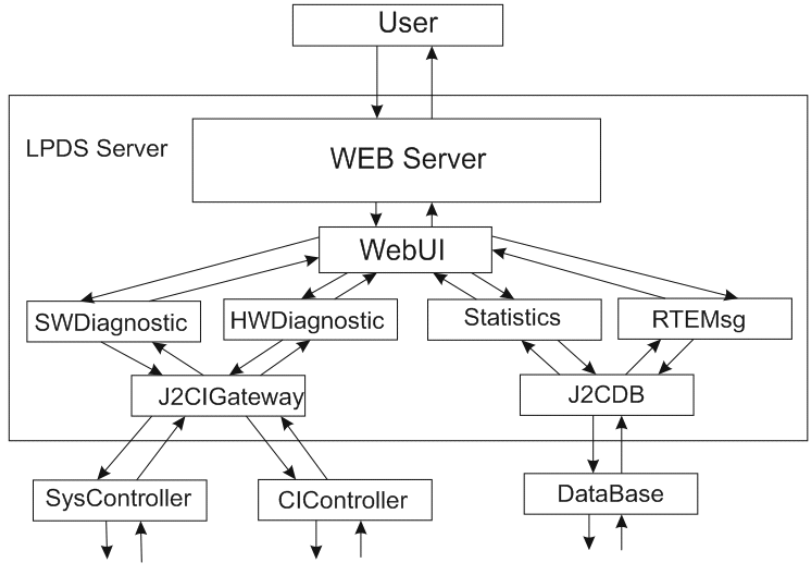


Fig. 3. Components of LPDS.

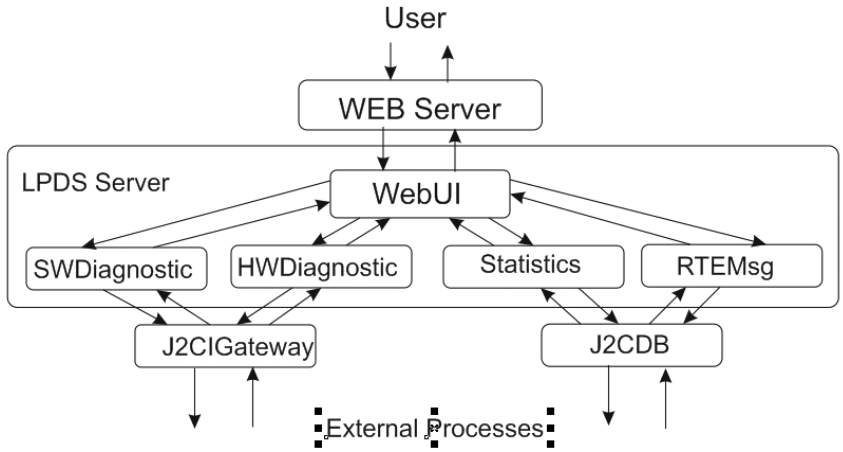


Fig. 4. CSP processes after the decomposition.

HW and OS diagnostics (HWDiagnostics and SWDiagnostics), statistics and special run-tie error messages components (Statistics and RTEMsg), as well as some gateways (J2CIGateway and J2CDB) that transform messages from internal to external components.

After CSP decomposition the LPDS was transformed to the CSP processes –given on Fig. 4. The point-to-point channels between the processes, shown with arrows, describe the communication paths. The names of the cannels describe the ways of messages exchange between two components - for example WebUItoSWDiag or SWDiagtoWebUI, etc. The channel names are not provided on the figure.

After analysis of customer requirements and the priorities of the exchanged messages and data between external and internal processes (components) the following composition of the processes in LPDS system was proposed

$$LPDS\_System = (J2CIGateway \,||\, J2CDB) \overleftarrow{\|} (SWDiagnostics \,||\, HWDiagnostics) \overleftarrow{\|} (Statistics \,||\, RTEMsg) \overleftarrow{\|} WebUI$$

As can be seen the gateway processes need highest priorities, next are the diagnostics processes followed by statistics and RTE messages processes. The lowest priority has the WEB UI process.

Each of the gateway processes was mapped to a single OS process. Other processes were implemented as threads inside one OS process. The threads are grouped with 3 levels of priorities: highest for SW and HD diagnostics, midium for statistics and RTE mes-

sages and lowest for WEB UI. So, the LPDS system takes the following form:

LPDS_System = J2CIGateway || J2CDB $\overset{<-}{\|}$ LPDS_Subsysten

where:

LPDS_Subsystem = (SWDiagnostics || HWDiagnostics) $\overset{<-}{\|}$ (Statistics ||RTEMsg) $\overset{<-}{\|}$ WebUI

In the beginning all processes, including the WEB server one, were allocated on one computer. After off-line analysis of log files and the performance of the LPSD system the decision to move the WEB server to another computer was taken and all other LPDS processes were left to the first computer. The decision was based on the overload of the computer because of highest communications among the WEB server and up to 60 simultaneously connected users (browsers), supported by it.

The described above algorithm was used for refining the software architecture of an Automatic Number Plate Recognition system /ANPR/ [4, 5] by mapping efficiently the CSP processes to corresponding OS tasks and threads. It increased the performance of the entire ANPR which led to the opportunity to register speed violence of over 20 % in km/h, higher than before the application of the algorithm. Next improvement was obtained on the same ANPR system, equipped with two cameras (infrared and color). Usually, the maximum car speed registered by two cameras does not exceed 130 km/h. After the application of the CSP process-oriented approach and the mentioned algorithm, the speed increases to 160 km/h without any hardware changes.

Application of the algorithm in the automated identification system (ASI) [6] intended to optimize product prices in the supermarkets significantly reduced the execution time of the entire ASI. This result was obtained by splitting the calculation over clustered in categories products and running the calculations (processes) in parallel.

CONCLUSIONS

The correctness of the algorithm was proved in the development of a number of real-time systems intended to parcels' and letters' processing, and in the discussed above system. Usually, these systems include up to 10-20 OS processes with up to 10 threads per process. More details regarding the use of the process-oriented CSP approach and the algorithm for the elaboration of these systems are given in [7, 8].

The algorithm is effective when developing complex software systems from scratch. It is not effective or not applicable when building systems with third party executable processes which can not be recompiled or rebuild. Then redistribution or reallocation of threads from one process to another is impossible, as well as merging or splitting of processes. A possible solution in this case is to use the Windows Performance Monitor program or similar ones and their counters, in order to take decisions for processes' allocation/reallocation over processors or cores.

REFERENCES

1. C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985, extended and updated 2004.
2. G. Hilderink , Embedded Control System Software Design based on CSP, TESI Workshop 2002 - paper, University of Twente, The Netherlands, 2002.
3. A. Atanassov, Web-Based System for Monitoring, Diagnostics and Control, of Postal Automation Systems, International Conference Automatics and Informatics'11, proceedings, published by John Atanasoff Society, of Automatics and Informatics, Bulgaria, Sofia, October 3-7, 2011.
4. A.Atanassov, Advanced Software Architecture Of an Automatic Number Plate Recognition System, J. Univ. Chem. Technol. Met. (Sofia), **47**, 1, 2012, 46-53.
5. A. Atanassov, Controlling An Automatic Number Plate Recognition System Via Web-Based Component, International Conference Automatics and Informatics'10, proceedings, published by John Atanasoff Society, of Automatics and Informatics', Bulgaria, Sofia, October 3-7, 2010.
6. A. Efremov, A. Atanassov, F. Tomova, D, Efremova, Combined group step regression and its application in market segment, Journal Automatics & Informatics volume 1/2012, Sofia, 2012.
7. A. Atanassov, Process-oriented Approach to Control Interface Communications of Automated Parcels' Processing Systems, Journal Automatics & Informatics volume 4/2009, (pp 46-53), Sofia, 2009.
8. A. Atanassov, F. Tomova, Web-Based Subsystem For Tuning Of An Automatic Plate Number Recognition System, Sixth International Conference, Challenges In Higher Education And Research In 21st Century, 2008, (pp 399-402), Sozopol, Bulgaria, 2008.