BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **9**, No 1

Sofia • 2009

Models of Autonomous Control Systems Software

Plamen Hristov, Plamen Angelov, Mariana Gousheva

Space Research Institute, 1000 Sofia, 6 Moskovska Str.
E-mail: phristov@space.bas.bg

Abstract: *Some models of the autonomous spacecraft computer systems are discussed in the paper. The ways to reach the autonomy are described – high reliability and adaptability of the system, reached by application of Artificial Intelligence elements, formal structural models (specification, verification and real time control) and dynamic reconfiguration [1, 2, 4, 9, 10, 11].*

A method using trace models in real time for testing of system operations is described. The method is suitable for systems specified by process algebras (CSP – Communicating Sequential Processes [3], Timed CSP [6], and ASM – Abstract State Machine [5], and TAM – Temporal Agent Model [7]), where process traces are used. The computing processes and these in the hardware are observed.

Special control processes (tracers) are specified. They allow to control in real time the correspondence of the real system traces and preliminary computed traces (the formal specification) and send messages to the other subsystems when no correspondence (e.g. to specification and dynamic reconfiguration subsystems).

The method allows finding in real time the software and hardware incorrectly functioning (disparity of the traces) based on rigorous mathematical specification.

In the paper are described: an autonomous control system structural model, an algorithm for real time traces control, possibilities for implementation, using OCCAM or PC libraries – CPPCSP – C^{++} Communicating Sequential Processes library [8], JCSP.

The application of trace models in real time allows to find any system incorrectly operating and this is one of the ways for increasing the onboard computer systems' reliability and to achieve full operation autonomy.

Keywords:*CSP models, autonomous control system, spacecraft, sofware.*

I. Introduction

The autonomy of the onboard computer control and information systems becomes more and more important with the deep space unmanned flights. It means higher reliability of the system and possibility to change system configuration in real-time depending of the system and environment state. How to reach this?

A new generation of spacecrafts with a new generation of control systems is needed for these missions. Their most important specialty is the possibility for long autonomous operation and others:

- minimal dependence from the ground systems;
- operation by general commands;
- planning and scheduling;
- high reliability of the system hardware and software;
- system dynamic recovering and reconfiguration.

These principles are most fully realized in the Remote Agent (RA) model based system [4].

The Aerospace Control Systems (ACS) for aircraft or spacecraft navigation are complex multiprocessor systems. They include many hardware and software objects that are communicating in a real-time mode. The standard stabilization and navigation systems are not fault resistant and they have static configurations and insufficient reliability. The most important characteristic of these systems is their reliability. As NASA reports many crashes were not due to hardware but software failure [9]. To increase the ACS software reliability formal methods are used in recent years [10, 11].

Usually formal methods implementation consists of two steps: specification and verification. The particularity of the approach described in the paper is that it includes in addition a real-time trace control which allows relatively easy computer realization of software specification and verification. The approach also includes method for dynamic reconfiguration of ACS structure in accordance with the developed formal model.

This paper presents the author's approach for creating of autonomous control and modeling system called ACS (Autonomous Control System). The approach is characterized by:

Using of formal models and methods at all the system operating cycle;

The system is presented as a set of objects freely configurable and connectible by channels [2];

Artificial Intelligence (AI) is used in the stage of system reconfiguration or as a part of the control process.

The main differences of ACS from RA are:

ACS executes control and development functions – in real time it makes dynamic reconfiguration (analysis, synthesis, etc.) of the execution control system;

AI functions are used in the control process if it is necessary and in the system development.

Formal models and methods are used repeatedly – to create and verify formal specification and to control the traces of the system processes, all in real time.

The structure of the ACS system is shown in Fig. 1.

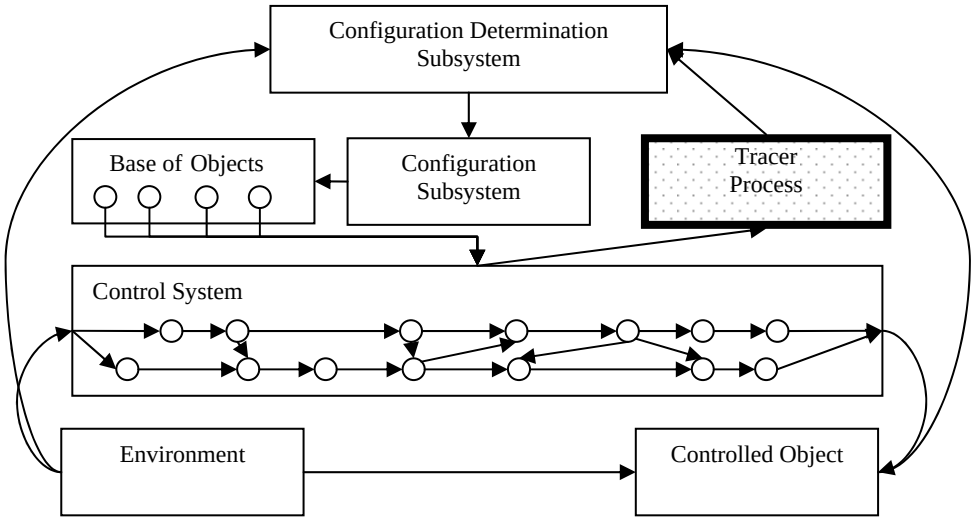

Fig.1. The conception of ACS as set of objects and subsystems

II. ACS description

The approach includes:

1. Software formal models (specifications). The software formal models are a process models type (a sequential process, that can be executed parallel), where the processes are communicating (and are synchronized) by a message exchange.
2. Software system verification, using Hoare's (CSP) specifications and laws. With the CSP theory the software properties can be specified and verified.
3. System operations control, using Hoare's trace models. The traces describe all events that have place, or can have place in the computing process, and can be used for run-time control.
4. System structure and parameters run-time determining on the base of information about control system object and background states. This is a separate subsystem of the computer control system. All algorithms may be used here.
5. Dynamic reconfiguration: on the base of the specification, the system structure and parameters are implemented in the general software model frame.

Corresponding to this approach the ACS is treated like a set of software and hardware objects that can be configured to exchange information and to synchronize and get access to common system resources.

The CSP models describe the functions and structure of the system by a set of communicating sequential processes, that can be executed parallel. The models include CSP-description, alphabet, trace and specification. The interactions are synchronous by one-direction channels.

The models described below represent a RTS system that consists of four main processes:

(CONTROL, MEASUREMENT, SIMULATION, MODEL):
MODELING = CONTROL || MEASUREMENT || SIMULATION || MODEL,

where the symbol || indicates parallel processes.

The alphabet of the processes determines a set of events, logically possible for the parallel system:

α(CONTROL || MEASUREMENT || SIMULATION || MODEL) = αCONTROL $\cup$ αMEASUREMENT $\cup$ αSIMULATION $\cup$ αMODEL,

where α is process alphabet.

The traces of the parallel processes are determined by:

trace(CONTROL || MEASUREMENT || SIMULATION || MODEL)={t | (t $\uparrow$ αCONTROL) $\in$ trace(CONTROL) & (t $\uparrow$ αMEASUREMENT) $\in$ trace(MEASUREMENT) & (t $\uparrow$ αSIMULATION) $\in$ trace(SIMULATION) & (t $\uparrow$ αMODEL) $\in$ trace(MODEL) & t $\in$ (αCONTROL $\cup$ αMEASUREMENT $\cup$ αSIMULATION $\cup$ αMODEL)},

where trace is a process protocol, describing the events, which the process has gone to this moment. The symbol $\uparrow$ indicates shrink of the trace on the some set, such as the alphabet.

Loop process, describing the system functions by separate loops:

LP_{i-}= {PR, CH, PAR, ARG}, where: PR – procedure, implementing the process functions; CH – set of information channels; AGR – set of aggregates; PAR – parameters.

CSP model is:

LoopProcess = c[0]?x $\rightarrow$ AgregatesList; c[n]!y $\rightarrow$ ENDprocess; ModelProcesses,

where: c[i] indicate channels with numbers; AgregatesList – linear aggregates list;

?x – CSP procedure to receive a value from channel c and assumes it to the x variable;

!y – CSP procedure to transmit the value of y variable to the channel;

ModelProcesses – modeling processes.

The traces are:

LoopProcess=P;Q;ModelProcesses;
P=c[0]?x$\rightarrow$AgregatesLIst; Q=c [n]!y$\rightarrow$ENDprocess;
trace(P) = {t|t=<>V(t_0=c[0]?x & t'$\in$ trace(AgregatesList))} = {<>} U {<c[0]?x>$\wedge$t|t $\in$ trace(AgregatesList)};
trace(Q)={t|t=<> V (t_0 = c[n]!z & t'$\in$ trace(ENDprocess))} = {<>} U {<c[n]!z>$\wedge$t|t $\in$ trace(ENDprocess)};

trace(LoopProcess)= {s;t|s $\in$ trace(P)& t $\in$ trace(Q); r|s $\in$ trace(P)& t $\in$ trace(Q)& r $\in$ trace(ModelProcesses)}.

The symbol ^ indicate after (between traces); t_0 – beginning of the trace; t' – end of the trace; <> – empty trace.

A list of aggregates CSP model is:

AgregatesList = (c[i]?z → AGREGATE[i]; AgregatesList | c[i]!z → AGREGATE[i]; AgregatesList) | (next → AGREGATE[i]; AgregatesList) | (end_list → ENDprocess),

where END process is a special process, the alphabet of which has only one event, that indicate successful end.

AGREGATE=input(x) → AgrTransferFunction(x:y); output(y) → ENDprocess;
AgrTransferFunction = (f_1→ENDprocess) | (f_2→ENDprocess) |...| (f_n→ENDprocess);

where f1,...,fn – functions, doing the transformation x→y.

There are two traces:

AgregatesList

(c[i]?z → AGREGATE[i]) = P1; (c[i]!z → AGREGATE[i]) = P2 ;
(next → AGREGATE[i] = P3;(end_list→ENDprocess) = P4 ;

trace(P1)={<>} U {<c[i]?z>^t|t ∈ trace(AGREGATE[i])};
trace(P1;AgregatesList)={s;t|s ∈ trace(P1) &t ∈ trace(AgregatesLIst)};

trace(P2)={<>} U {<c[i]!z>^t|t ∈ trace(AGREGATE[i])};
trace(P2;AgregatesList)={s;t|s ∈ trace(P2) &t ∈ trace(AgregatesLIst)};
trace(P3)={<>} U {<next>^t|t ∈ trace(AGREGATE[i])};
trace(P3;AgregatesList)={s;t|s ∈ trace(P3) & t ∈ trace(AgregatesLIst)};
trace(P4)={<>} U {<end_list>^t|t ∈ trace(ENDprocess)};
trace(P4;AgregatesList)={s;t|s ∈ trace(P4)&t ∈ trace(AgregatesLIst)};
trace(AgregatesList)= {t|t=<> V (t0 ∈ B & t' ∈ trace($P(t_0)$)} ,

where B = {c[i]?z,c[i]!z,next,end_list}; $P(t_0)$= (P1|P2|P3|P4).

– AGREGATE
P = (input(x)→ AgrTransferFunction);
Q = (output(y)→ENDprocess); AGREGATE=(P;Q);
trace(P)={<>} U {<input(x)>^t|t ∈ trace(AgrTransferFunction)};
trace(Q)={<>} U {<output(y)>^t|t ∈ trace(ENDprocess)};
trace(AGREGATE)= {s;t|s ∈ trace(P) & t ∈ trace(Q)}.

– AgrTransferFunction
trace(AgrTransferFunction)= {t|t=<> V (t_0 ∈ B & t'∈ trace($P(t_0)$)},

where B = {f1,f2,.., fn} alternative

$P(t_0)$=f1→ENDprocess|f2→ENDprocess|...| fn→ENDrocess);

In a similar way the other system objects are defined.

III. System operations control method based on the CSP trace models

The CSP theory [3] proposes appropriate means for preliminary specification of the system functions and structure, and for the following operation control. These means are: specifications, alphabets and traces of the processes.

The general algorithm of the control method (Fig. 2) includes:

the software specification is defined, including the set of system objects;

the program system implementation and verification (CSP specifications verification method [1, 2] – function Sat (P, S));

all possible traces computing for the specified system (procedure trace (P));

the system start;

the special process (tracer) observes and registers the system traces and determines if they are valid – function IsTrace(s, P);

When invalid trace is registered – emergency situation and message to the configuration subsystem.

The function SatP,S) that decides whether the system implementation satysfies the system specification:

```
Sat(P,S(np)) =
if P = STOP then          if np = <> then return TRUE;
else
    return FALSE;         end;
  elsif P sat S(np) & (c ∏ P) then
     if (np = <> V (np0 = c & S(np'))) then return TRUE;
       else return FALSE;          end;
  elsif P sat S(np) & (c  ∏ d ∏ P) then
    if (np ≤ <c,d> V (np ≥ <c,d> & S(np")))
      then return TRUE; else return FALSE;
elsif P sat S(np) & Q sat T(np) & (c∏P|d∏Q)
  then
  if (np = <> V (np0 = c & S(np')) V (np0 = d & T(np')) /*S(np') & T(np') are the
specifications of the chosen alternative */
then return TRUE;
else return FALSE;
elsif ∇ x ∈ B.(P(x) sat S(np,x)) & (x:B ∏ P(x)) then
if (np = <> V (np0 ∈ B & S(np',np0)))
then return TRUE; (*a process with choice from a set*)
else return FALSE;
elsif P sat S(np) & Q sat T(np) & P || Q then      if (P||Q) sat (S(np |‘ ∝P) & T(np |‘
∝Q)) then return TRUE; (* parallel processes *)
else return FALSE;
———  other standard structures ——-
else return FALSE;       End;
———  other standard structures ——
```

else return FALSE;
End;

The function trace(P) that computes the system traces:

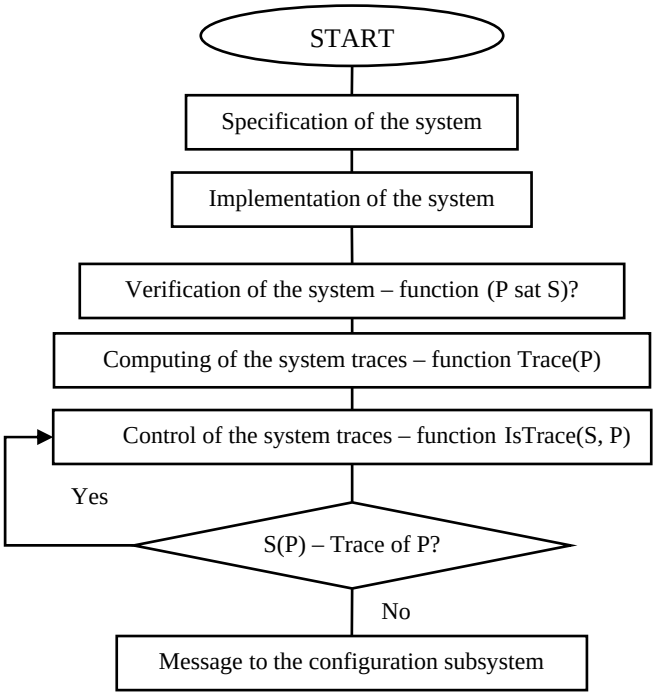

Fig. 2. The Method for Control of the System Traces

trace(P)= (μR)* &
if P = ($STOP_P$) then trace(P)= \<>\;
elsif P = (c→ P) then trace(P)= \<> U \<c> ∧ t|t ∈ trace(P)\;
elsif P = (c → P | d → Q)then trace(P)= \t|t = <> V (t_0 = c & t'∈ trace(P)V (t_0 = d & t'∈ trace(Q))\;
elsif P = (x:B → P(x)) = \t|t = <> V (t_0 ∈ B & t'∈ trace(P(t_0)))\
elsif P = (mX : A.F(x))) = U trace($F^n(STOP_A)$);
elsif P = (P || Q)then trace(P)= trace(P)Ç trace(Q);
else t = trace(P||Q) then
trace(P||Q)=\t|(t |`μR) ∈ trace(P)& (t |`μQ) ∈ trace(Q)& t ∈ (aR U aQ)*\
..... other structures.....
else
end.

The function IsTrace(s,p) that decide whether the trace s is valid for the process P:

```
IsTrace(s,P)=
        if s = NIL then TRUE;
                elsif P(s0) = "BLEEP then FALSE;
                else IsTrace(s', P(s0));
end.
```

IV. Configuration subsystem

It implements system configuration in real-time (dynamic reconfiguration) by:

1) creating instances (objects) of system classes in the system memory;
2) connecting objects by determining channel numbers.

V. Implementation

These models can be implemented in several ways:

In a transputer environment, using OCCAM

In standard PC and similar environments, using Java (JCSP library) or C^{++} (C^{++}CSP library) [8].

VI. Conclusion

This is an integrated approach for the design and study of aerospace control systems based on CSP models. This approach is implemented by the methods for specification, verification and trace control. The models describe in general the functions of the system by using a rigorous mathematical theory.

The control and verification methods allow the creation of high-reliability systems by preliminary verification and run-time system observation. The control method, based on a trace model, is new in this type of systems. The main advantage of the approach is the completeness and the possibility to achieve a certain degree of assurance in the system quality.

Dynamic reconfiguration method allows the system to be implemented in multi- or single processor environments.

References

1. Hristov, P. L., P. S. Angelov. Formal Methods used in Development and Real Time Simulation of Aerospace Control Systems. – International Journal "Information Theories and Applications", Vol. **9**, 2003, No 5, 174-180.
2. Hristov, P. L., P. S. Angelov. A Model of Autonomous Control System for Deep Space Missions. – In: Proc. of International Conference on Recent Advances in Space Technologies RAST'2003, 20-22 November, 2003, Istanbul, Turkey, 368-372.
3. Hoare, C. A. R. Communicating Sequential Processes. London, Prentice Hall International, UK, LTD, 1985.
4. Douglas, B., G. Dorais, E. Gamble, B. Kanefsky, J. Kurienà, G. K. Man, W. Millar, N. Muscettola, P. Nayak, K. Rajan, N. Rouquette, B. Smith, W. Taylor, Y.-W. Tung. Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment. AIAA-99-4512. **http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/17741/1/99-1183.pdf**

5. Huggins, J. K., C. Wallace. An Abstract State Machine Primer. Technical Report CS-TR-02-04, Computer Science Department, Michigan Technological University, 4 December 2002.
http://www.kettering.edu/~jhuggins/papers/primer.pdf
6. Reed, G. M., A.W. Roscoe. A Timed Model for Communicating Sequential Processes. – In: Proc. of ICALP'86, Springer, LNSC 226, 1986.
7. Lowe, G. Infinite Behaviours in the Temporal Agent Model. PRG-R-3-95. Oxford University Computing Laboratory, Oxford **OX1 3QD**.
ftp://ftp.comlab.ox.ac.uk/pub/Documents/techreports/TR-3-95.ps.gz
8. Brown, N., P. Welch. An Introduction to the Kent C^{++}CSP Library. Communicating Process Architectures'2003, IOS Press, 2003, Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, England.
9. Why Are Formal Methods Necessary? NASA Larc Formal Methods Program, 2001.
http://shemesh.larc.nasa.gov/fm/
10. Ben, L., Di Vito. Formalizing New Navigation Requirements for NASA Space Shuttle. 30 Research Drive, Hampton, Virginia, 1999.
https://eprints.kfupm.edu.sa/41798/1/41798.pdf
11. Fung, F., D. Jamsek. Formal Specification of a Flight Guidance System. NASA/CR-1998-206915, 1987.
http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19980031982_1998093530.pdf