Modelling a Fault-Tolerant Distributed Satellite System

Kashif Javed
Turku Centre for Computer Science (TUCS)
Department of Information Technologies
Abo Akademi University
Turku, FIN-20520, Finland
Kashif.Javed@abo.fi

Elena Troubitsyna
Department of Information Technologies
Abo Akademi University
Turku, FIN-20520, Finland
Elena.Troubitsyna@abo.fi

***Abstract*— Ensuring correctness of a complex distributed and mode-rich collaborative satellite system is a challenging task that requires formal modeling and verification. In this paper, we propose a model of a distributed Attitude and Orbit Control System. Mode transitions in such systems are governed by a sophisticated synchronization procedure. We demonstrate how to model and verify such a procedure in order to ensure mode consistency.**

Keywords-distributed mode-rich systems; satellite software; fault tolerance; synchronization

I. INTRODUCTION

Behavior of satellite systems is often structured in terms of modes. Modes – mutually exclusive sets of system behavior define different functional profiles of the system [4,5]. An important problem associated with designing mode-rich satellite systems is to ensure correctness of mode transitions.

In this paper, we propose an approach to modeling and verification of distributed Attitude and Orbit Control System – D-AOCS [1,2]. D-AOCS is a typical example of a mode-rich collaborative system. It consists of two independent mode managers that should negotiate and coordinate their actions. Collaboration between mode managers is not trivial – faults of components might prevent the mode managers from following the agreed course of actions. As a result new negotiations would be initialized to achieve synchronization under the new conditions.

The proper synchronization is paramount for ensuring mode consistency. In general mode consistency can be seen as a high-level guarantee of a proper functioning of a distributed system deployed on the space craft. The complex collaboration procedure precedes each mode transition step.

We demonstrate how to model and verify handshaking protocol ensuring that modes are changed consistently. An important part of our modeling is fault tolerance. We demonstrate how to ensure consistency of not only nominal but also backward mode transitions, i.e., transitions to the degraded modes that are responsible for error recovery. The novelty of the proposed approach is in treating fault tolerance of collaborative systems as a problem of ensuring mode consistency.

Section II explains the state-of-the-art of AOCS structure. Section III presents AOCS architecture covering unit manager, mode manager, and fault tolerance. Handshake protocol is explained in detail in Section IV and the proposed system design using handshake is discussed in Section V. Finally, Section VI provides a brief summary of conclusions and future work.

II. STATE-OF-THE-ART STRUCURE

Attitude and Orbit Control System (AOCS) is extensively used in the design and development of modern satellites. The major objective of an AOCS is to ensure controlled movements of the satellites in order to maintain required attitude and remain in the given orbit. As disturbance of the atmosphere tends to change orientation of the satellites, there is a serious need to continuously control and monitor its attitude. A number of sensors are employed to collect data for the purpose of controlling attitude. Appropriate corrective measures are taken by the actuators to keep the right path and orbit whenever there is change detected in the data sent by the sensors. This requirement is very essential for supporting needs of payload instruments as well as for the fulfillment of satellite's mission.

The top level schema of an AOCS is shown in Figure 1.

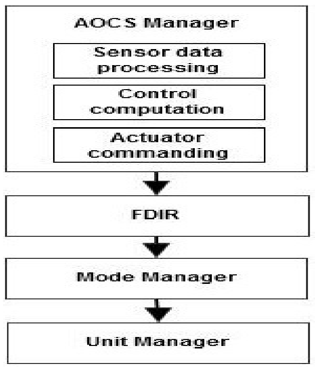

Figure 1: Top Level Schema of AOCS

The AOCS manager consists of three components (i.e., sensor data processing, control computation and actuator commanding). Control computation part handles all the data and measurements using state-of-the-art control algorithms and gives commands to the actuators for ensuring correct path and attitude. Different types of controllers are required for completion of specific mission stages. Normally, two control algorithms are used during the operational mode of the satellites.

Each unit of the satellite has a unique status (i.e., free, reserved, or locked) for its usage while avoiding conflicts during reconfiguration [10]. An actuator, payload or sensor remains free when it is idle in any mode. The reserved status means that a sensor/actuator/payload is to be used shortly but it is not yet ready. When any unit is allocated and is being used for its required operation, then it is turned into the locked status.

III. ARCHITECTURE

In this paper, we consider a distributed version of Attitude and Orbit Control System. Attitude and Orbit Control System (AOCS) [1] is a generic component of a spacecraft. Behavior of AOCS is structured using the notion of modes – mutually exclusive sets of system behavior. The complexity of designing distributed AOCS lies in the fact that mode management is decentralized, i.e., it is performed by several mode managers. Distributed AOCS (D-AOCS) has a complex architecture. It consists of AOCS Manager, Unit Manager, Several Mode Managers and FDIR (Failure Detection, Isolation and Recovery) Manager. AOCS Manger deals with two controllers -- Control Pointing Controller (CPC) and Fine Pointing Controller (FPC). The purposes of CPC and FPC are to direct line of sight as well as to provide coarse and fine accuracy. Unit level state transitions and mode transitions are managed by Unit Manager and Mode Manager respectively. FDIR Manager ensures handling of branch state transition errors and controller phase transition errors [2]. Two managers -- Mode Manager 1 (MM1) and Mode Manager 2 (MM2) are responsible for the global mode logic of D-AOCS. The architecture of Unit Manager and Mode Managers is described below.

A. *Unit Manager*

The Unit Manager in D-AOCS organizes the internal states of the units. The components of Unit Manager are supervised by the Mode Manager. The controlled units include Earth Sensor (ES), Sun Sensor (SS), Star Tracker (STR), Global Positioning System (GPS), Reaction Wheel (RW), Thruster (THR) and Payload Instrument (PLI). All unit components are responsible for mode synchronization, decision making on unit states, performing branch state transitions and unit reconfiguration [4,5]. SS, STR, GPS, RW and PLI provide data to the AOC Manager. RW and THR execute the commands from AOC Manager. These units are also responsible for detection and reporting the branch state transition errors [1].

Every unit consists of two identical branches -- the nominal and redundant ones. At any instance of time only one branch is active. A unit branch in the 'on' state is always assigned locked status and the unit branch in 'off' state has unlocked status. There are six states of unit components -- on, off, coarse, fine, standby and science. The internal states of ES, SS, STR, RW and THR are either 'on' or 'off'. Three possible GPS's operational states are 'off', 'coarse' and 'fine'. PLI's state can be in 'off', 'standby' or 'science' [3].

B. *Mode Managers*

The global mode transitions are managed by the two mode managers -- MM1 and MM2. Each mode manager's controls different units. Each mode manager is responsible for checking the preconditions of mode transitions, managing the controllers and the units, and initiating and completing the mode transitions. The global modes are correspondingly Off, Standby, Safe, Nominal, Preparation, and Science [10]. Below we give a brief description of each mode:

Off: After the central data management unit completes booting of AOCS software, the satellite instantly goes into the off mode.

Standby: The process of separation of the satellite from the launcher is monitored during the standby mode.

Safe: After successful separation from the launcher, the satellite switches to the safe mode. The satellite obtains a stable attitude and the CPC is activated.

Nominal: After transition to this mode, FPC is activated, while CPC is switched off. PLI is actoviated to provide measurements for FPC.

Preparation: FPC is achieved in the preparation mode and PLI gets ready to perform the necessary tasks.

Science: PLI carries out the required tasks and stays in science mode till the desired tasks are completed.

MM1 and MM2 communicate with each other to synchronize on mode transitions that are performed in parallel. Let us describe the scenario of mode transitions. After a mode transition to off or standby is done, every unit branch goes to off state and both controllers are idle. After that, both mode managers communicate with each other. If there is no error then transition to the next mode is executed. When the mode is switched to safe, the selected branches of ES, SS and RW are turned to 'on' state and only FPC remains idle. Both mode managers send messages to inform each other that no error occured in the given modes. After a handshake, they perform the mode transition to the nominal mode. In a mode transition to the nominal mode, the required branches of RW, STR and THR are put to the 'on' state and GPS is put into the'coarse' state. The messages sent and received by the mode managers notify each mode manager that no unit or controller error has occured. Then the preparation mode is reached, the concerned branches of RW, STR & THR are in the 'on' state and GPS & PLI are in the 'fine' state and 'standby' state respectively. They ensure the correctness of the modes in MM1 and MM2 and make a transition to the science mode. In case of the science mode, the preffered branch of PLI operates with 'science' state. All other units keep their previous state. When a mode transition goes to nominal, preparation or science mode, only CPC remains idle. MM1 and MM2 both inform each other regarding success of mode transition.

C. *Fault Tolerance*

Fault tolerance aims at providing the system with the means to continue its function in spite of errors of its components. In the D-AOCS backward error recovery is adopted, i.e., if an error occurs, the system gets back to some previous state to handle the error. The roll back error recovery is implemented by the backward mode transitions. The mode roll-back depends on branch state transition errors and phase transition errors.

There are different aspects relating to the branch state transition errors. When a branch state transition error on the redundant branch of ES, RW or SS occurs and there is no error in the remaining redundant branches, then the mode goes back to off mode. If the redundant branch of GPS, STR or THR gets corrupted, it results a mode transition to safe. A mode transition to nominal takes place when there is a branch state transition error on the redundant branch of PLI.

The important error checks are incorporated to deal with the attitude or phase transitions. When the current mode is safe and a non-negligible phase error is produced, it results in a mode transition to off. If the phase error is generated in the nominal, then it goes back to safe. In case the existing mode is preparation and a phase error occurs, a mode transition to nominal takes place. A mode transition to preparation takes place when a phase error occurs in the science mode [3].

In case of unit reconfiguration, a branch state transition error on the nominal branch of any unit causes a unit reconfiguration if there is no branch state transition error on the redundant branches of that particular unit.

If the mode task is not completed within a given time interval or multiple errors occur in the unit branches and controller phases, then timeout signal is produced for safe condition.

IV. HANDSHAKE PROTOCOL

Handshaking is a process in which connection is established among two processes and information is transferred from one process to another without the need for human involvement to set constraints. MM1 and MM2 do handshake with each other to update the condition of their modes. Different scenarios of handshake protocol are explained covering the following key points:

If all conditions of unit states and controller phases within each mode of MM1 and MM2 fulfill their requirements, then mode managers pass the ‘no error’ message to notify that the mode is in the error-free state. It results in the forward mode transition, i.e., the mode manager switches the current mode to the next mode as described in Section III.

If an error occurs during a mode transition of MM1 and there is no error in the mode of MM2, then MM1 sends an ‘error’ message to MM2. MM1 executes error recovery, i.e., starts backward mode transtion according to the Section III. Until the error recovery of MM1 is not completed, MM2 keeps on waiting. After the successful error recovery, both mode managers proceed to the next mode.

When an error occurs only in the mode of MM2, then MM1 receives an ‘error’ message from MM2. MM1 waits until error has been recovered in MM2. The mode managers switch to next mode after receiving the information from MM2 that the error is recoverd.

Upon receiving an ‘error’ message from MM1 and MM2 simultaneously, error recovery starts in both mode managers as mentioned in Section III. The backward mode transitions are executed in MM1 and MM2. After achieving the successful recovery, mode managers move to the next mode.

There are two types of errors -- the unit branch state transition errors and controller phase transition errors. Handshaking algorithm for handling such type of errors is quite complex as specified below:

```
void handshake(int u_MM1, int u_MM2,int c_MM1,int c_MM2) {
// ‘u’ denotes unit error flag and ‘c’ denotes controller error flag
 if (u_MM1==0&&u_MM2==0&&c_MM1==0&&c_MM2==0) {
     /* The associated code illustrates that no error occurs in the unit
     branchs of ES, SS, RW, GPS, STR, THR or PLI and controller
     phase of CPC or FPC in the given mode of MM1 and MM2. It
     accounts the forward mode transition according to the Section
     III. */}
elseif(u_MM1==1&&u_MM2==0&&c_MM1==0&&c_MM2==0) {
     /* The associated code illustrates that an error occurs in the unit
     branch of ES, SS, RW, GPS, STR, THR or PLI in the given
     mode of MM1. It accounts the backward mode transition
     according to the Section III.  MM2 stays on waiting until an
     error is recovered. */}
elseif(u_MM1==0&&u_MM2==1 &&c_MM1==0&&c_MM2==0) {
     /* The associated code illustrates that an error occurs in the unit
     branch of ES, SS, RW, GPS, STR, THR or PLI in the given
     mode of MM2. It accounts the backward mode transition
     according to the Section III.  MM1 stays on waiting until an
     error is recovered. */}
elseif(u_MM1==1&&u_MM2==1&&c_MM1==0&&c_MM2==0) {
     /* The associated code illustrates that an error occurs in the unit
     branch of ES, SS, RW, GPS, STR, THR or PLI in the given
     mode of both mode managers. MM1 and MM2 account the
     backward mode transition according to the Section III. */}
elseif(u_MM1==0&&u_MM2==0&&c_MM1==1&&c_MM2==0) {
     /* The associated code illustrates that an error occurs in the
     controller phase of CPC or FPC in the given mode of MM1. It
     accounts the backward mode transition according to the Section
     III.  MM2 stays on waiting until an error is recovered. */}
elseif(u_MM1==0&&u_MM2==0&&c_MM1==0&&c_MM2==1) {
     /* The associated code illustrates that an error occurs in the
     controller phase of CPC or FPC in the given mode of MM2. It
     accounts the backward mode transition according to the Section
     III.  MM1 stays on waiting until an error is recovered. */}
elseif(u_MM1==0&&u_MM2==0&&c_MM1==1&&c_MM2==1) {
     /* The associated code illustrates that an error occurs in the
     controller phase of CPC or FPC in the given mode of both mode
```

```
        managers. MM1 and MM2 account the backward mode
        transition according to the Section III. */}
else {
        /* The associated code describes that it is an invalid condition.
        Program is terminated.*/}           }
```

V. Proposed System Design using Handshake

The proposed system design has been implemented using SystemC. SystemC can be used at system level for functional verification. The framework also supports event driven simulation environments [6]. It offers application program interface for transaction based verification, handling exceptions and verification tasks [7]. The system model consists of six defined modes named as A (Off), B (Standby), C (Safe), D (Nominal), E (Preparation) and F (Science). Three different operations have been implemented (i.e., forward mode transitions, backward mode transitions, and unit reconfiguration). The flow chart given in Figure 3 describes detailed design structure for only one transition from Mode E to Mode F of the system. When the system reaches to Mode E, it checks the error in the Mode E of both mode managers. Figure 3 shows the operations regarding error condition according to the scenarios and backward mode transitions according to the error types (Unit branch error (redundant/nominal) and controller phase error) they are discussed in Section IV and Section III respectively.

After necessary declarations of modes, units and controllers, the verification of the system are described in the following sections.

A. Verification of Forward Mode Transition

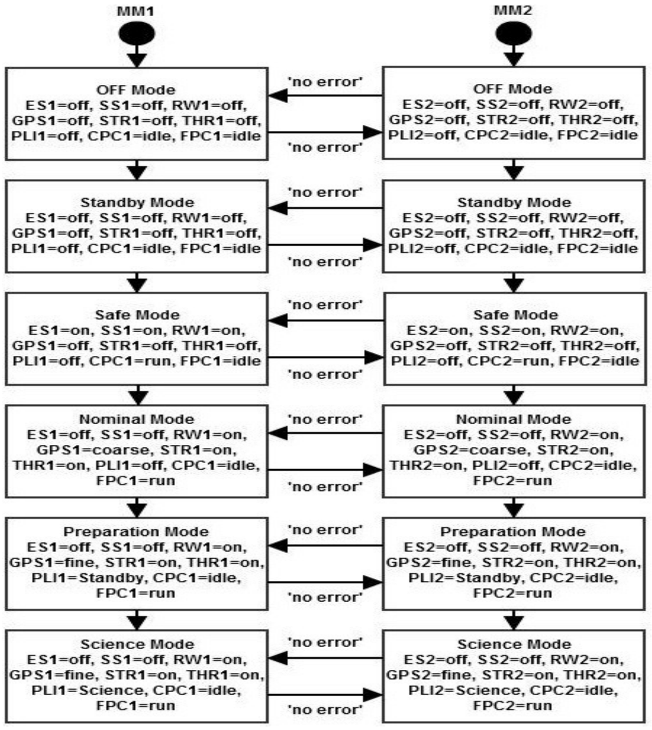
MM1
MM2
OFF Mode
ES1=off, SS1=off, RW1=off, GPS1=off, STR1=off, THR1=off, PLI1=off, CPC1=idle, FPC1=idle
OFF Mode
ES2=off, SS2=off, RW2=off, GPS2=off, STR2=off, THR2=off, PLI2=off, CPC2=idle, FPC2=idle
'no error'
'no error'
Standby Mode
ES1=off, SS1=off, RW1=off, GPS1=off, STR1=off, THR1=off, PLI1=off, CPC1=idle, FPC1=idle
Standby Mode
ES2=off, SS2=off, RW2=off, GPS2=off, STR2=off, THR2=off, PLI2=off, CPC2=idle, FPC2=idle
'no error'
'no error'
Safe Mode
ES1=on, SS1=on, RW1=on, GPS1=off, STR1=off, THR1=off, PLI1=off, CPC1=run, FPC1=idle
Safe Mode
ES2=on, SS2=on, RW2=on, GPS2=off, STR2=off, THR2=off, PLI2=off, CPC2=run, FPC2=idle
'no error'
'no error'
Nominal Mode
ES1=off, SS1=off, RW1=on, GPS1=coarse, STR1=on, THR1=on, PLI1=off, CPC1=idle, FPC1=run
Nominal Mode
ES2=off, SS2=off, RW2=on, GPS2=coarse, STR2=on, THR2=on, PLI2=off, CPC2=idle, FPC2=run
'no error'
'no error'
Preparation Mode
ES1=off, SS1=off, RW1=on, GPS1=fine, STR1=on, THR1=on, PLI1=Standby, CPC1=idle, FPC1=run
Preparation Mode
ES2=off, SS2=off, RW2=on, GPS2=fine, STR2=on, THR2=on, PLI2=Standby, CPC2=idle, FPC2=run
'no error'
'no error'
Science Mode
ES1=off, SS1=off, RW1=on, GPS1=fine, STR1=on, THR1=on, PLI1=Science, CPC1=idle, FPC1=run
Science Mode
ES2=off, SS2=off, RW2=on, GPS2=fine, STR2=on, THR2=on, PLI2=Science, CPC2=idle, FPC2=run
'no error'
'no error'

Figure 2: Forward Mode Transitions

When all the units are in off state, controller phases are in the idle phase, and no unit reconfiguration is in progress, then current mode is A in MM1 and MM2. The unit/controller error flag is set to low and mode managers exchange the information ('no error' message) of error-free mode status. After this, the mode moves forward to the next mode (i.e., Mode B) in MM1 and MM2. Hence, when all conditions of unit states and controller phases within each mode of each manager fulfill their requirements, mode managers update each other about the error-free mode conditions. Then the current mode switches to the next mode within each mode manager until it completes its operation after Mode F. Figure 2 illustrates the implemented procedure that corresponds to the forward mode transition for MM1 and MM2.

B. Verification of the Steps in the Backward Mode Transition

The backward mode transition depends on the two types of errors (i.e., unit branch state transition error and controller phase transition error). Handshaking procedure for handling these errors is given below.

1) Verification of the Steps in Unit Branch State Transition Error

Following part of the code segment describes the unit branch transition error in case of Mode E as shown in Figure 2. If there is an error in ES, SS or RW of MM1, MM1 switches to Mode A. If an error occurs in GPS, THR or STR of MM2, MM2 return to Mode C. However, if PLI gets an error in both mode managers, MM1 and MM2 both go back to Mode D. Before backward transition to the desired mode, the messages exchange information between the effected mode manager and the error-free mode manager to acknowledge the error status.

```
// Variable declarations
int FPC1,CPC1,FPC2,CPC2,u_MM1,c_MM1,u_MM2,c_MM2;
// unit states
const int off=0;const int on=1;const int coarse=2;
const int fine=3;const int unit=0;const int Standby=4;
const int Science=5;const int idle=0;const int run=1;
const int A=1;const int B=2;const int C=3;
const int D=4;const int E=5;const int F=6;
/* Each unit has two branches i.e., Nominal and Redundant,
here we deal with redundant branch of the units. */
int ES1,SS1,GPS1,STR1,RW1,THR1,PLI1; // MM1 Units
int ES2,SS2,GPS2,STR2,RW2,THR2,PLI2; // MM2 Units
    if(mode==E) {// Preparation Mode
    if((ES1!=off || SS1!=off || RW1!=on) && STR1==on &&
    GPS1==fine && THR1==on && PLI1== standby &&
    CPC1==idle && FPC1==run && ES2==off &&
    SS2==off && RW2==on && STR2==on &&
    GPS2==fine && THR2==on && PLI2== standby &&
    CPC2==idle && FPC2==run){
        u_MM1=1;c_MM1=0;
        u_MM2=0;c_MM2=0;
        /* The remaining part of the code, by calling the
        handshake protocol function on the basis of unit and
        controller error flag, is mentioned in Section IV.*/}
    else if(ES1==off && SS1==off && RW1==on &&
    STR1==on && GPS1==fine && THR1==on &&
PLI1==standby && CPC1==idle && FPC1==run &&
ES2==off && SS2==off && RW2==on && (STR2!=on ||
GPS2!=fine || THR2!=on) && PLI2==standby &&
CPC2==idle && FPC2==run){
      u_MM1=0;c_MM1=0;
      u_MM2=1;c_MM2=0;
      /* The remaining part of the code, by calling the
      handshake protocol function on the basis of unit and
      controller error flag, is mentioned in Section IV.*/}
else if(ES1==off && SS1==off && RW1==on &&
STR1==on && GPS1==fine && THR1==on &&
PLI1!=standby && CPC1==idle && FPC1==run &&
ES2==off && SS2==off && RW2==on && STR2==on
&& GPS2==fine && THR2==on && PLI2!=standby &&
CPC2==idle && FPC2==run){
      u_MM1=1;c_MM1=0;
      u_MM2=1;c_MM2=0;
      /* The remaining part of the code, by calling the
      handshake protocol function on the basis of unit and
      controller error flag, is mentioned in Section IV.*/}
else{
      /* The associated code describes that no transitions
      take place. */ }          }
else  cout<<" Program is terminated.";
```

2) Verification of the Steps in Controller Phase Transition Errors

When CPC and FPC do not fulfill the requirement of mode of any mode manager, the error flag is set to high and the affected mode manager is downgraded to previous mode after utilizing the handshake protocol by sending message to error-free mode manager. In case the phase of controllers in the given mode of both mode managers is corrupted, then both managers do the backward mode transition at once after acknowledging each other. The following portion of the code represents the scenario of phase transition for Mode E as illustrated in Figure 2.

```
//Variables are declared in the previous section.
if(mode==E) {// Preparation Mode
if(ES1==off && SS1==off && RW1==on &&
STR1==on && GPS1==fine && THR1==on &&
PLI1==standby && CPC1!=idle && FPC1==run &&
ES2==off && SS2==off && RW2==on && STR2==on
&& GPS2==fine && THR2==on && PLI2==standby &&
CPC2==idle && FPC2==run){
      u_MM1=0;c_MM1=1;
      u_MM2=0;c_MM2=0;
      /* The remaining part of the code, by calling the
      handshake protocol function on the basis of unit and
      controller error flag, is mentioned in Section IV.*/}
else if(ES1==off && SS1==off && RW1==on &&
STR1==on && GPS1==fine && THR1==on &&
PLI1==standby && CPC1==idle && FPC1==run &&
ES2==off && SS2==off && RW2==on && STR2==on
&& GPS2==fine && THR2==on && PLI2==standby &&
CPC2==idle && FPC2!=run){
      u_MM1=0;c_MM1=0;
      u_MM2=0;c_MM2=1;
      /* The remaining part of the code, by calling the
      handshake protocol function on the basis of unit and
      controller error flag, is mentioned in Section IV.*/}
else if(ES1==off && SS1==off && RW1==on &&
STR1==on && GPS1==fine && THR1==on &&
PLI1==standby && CPC1==idle && FPC1!=run &&
ES2==off && SS2==off && RW2==on && STR2==on
&& GPS2==fine && THR2==on && PLI2==standby &&
CPC2!=idle && FPC2==run){
      u_MM1=0;c_MM1=1;
      u_MM2=0;c_MM2=1;
      /* The remaining part of the code, by calling the
      handshake protocol function on the basis of unit and
      controller error flag, is mentioned in Section IV.*/}
else{
      /* The associated code describes that no transitions
      take place. */ }          }
else  cout<<" Program is terminated.";
```

C. Verification of the Steps in Unit Reconfiguration

If error exists on nominal unit branch at any mode of MM1 or MM2, then it is replaced by redundant unit branch in the given mode of mode manager. The unit reconfiguration is done to complete the remaining operation of the system. Unit reconfiguration is, however, a burden on the system and takes some time while switching from nominal branch to redundant branch of the unit. In case of the nominal unit branch in the given mode of both mode managers is corrupted, then unit reconfiguration is done in both mode manager after exchanging the information between the mode managers regarding unit reconfiguration.

The following piece of the code shows the scenario of unit reconfiguration for Mode E as shown in Figure 2.

```
//Variables are declared in the previous section. In reconfiguration
module, we also deal with nominal branch of the units. So, both
branches of the unit are declared separately.
//Nominal branches of MM1 and MM2
int N_ES1, N_SS1, N_RW1, N_GPS1, N_STR1, N_THR1, N_PLI1;
int N_ES2, N_SS2, N_RW2, N_GPS2, N_STR2, N_THR2, N_PLI2;
//Redundant branches of MM1 and MM2
int R_ES1, R_SS1, R_RW1, R_GPS1, R_STR1, R_THR1, R_PLI1;
int R_ES2, R_SS2, R_RW2, R_GPS2, R_STR2, R_THR2, R_PLI2;
if (mode==E) { //  Preparation Mode
if((N_ES1!=off || N_SS1!=off || N_RW1!=on) && R_ES1==off &&
R_SS1==off && R_RW1==on && N_ES2==off && N_SS2==off
&& N_RW2==on && R_ES2==off && R_SS2==off &&
R_RW2==on ) {
      u_MM1=1;c_MM1=0;
      u_MM2=0;c_MM2=0;
      /* The remaining part of the code, by calling the handshake
      protocol function on the basis of unit and controller error flag, is
      mentioned in Section IV.*/}
else if(N_GPS1==fine && N_STR1==on && N_THR1==on &&
N_PLI1==standby && R_GPS1==fine && R_STR1==on &&
R_THR1==on && R_PLI1==standby && (N_GPS2!=fine ||
N_STR2!=on || N_THR2!=on || N_PLI2!=standby) &&
R_GPS2==fine && R_STR2==on && R_THR2==on &&
R_PLI2==standby) {
      u_MM1=0;c_MM1=0;
      u_MM2=1;c_MM2=0;
      /* The remaining part of the code, by calling the handshake
      protocol function on the basis of unit and controller error flag, is
      mentioned in Section IV.*/}
else if((N_ES1!=off || N_SS1!=off || N_RW1!=on) && R_ES1==off
&& R_SS1==off && R_RW1==on && (N_ES2==off || N_SS2!=off
|| N_RW2!=on) && R_ES2==off && R_SS2==off && R_RW2==on
) {   u_MM1=1;c_MM1=0;
      u_MM2=1;c_MM2=0;
```

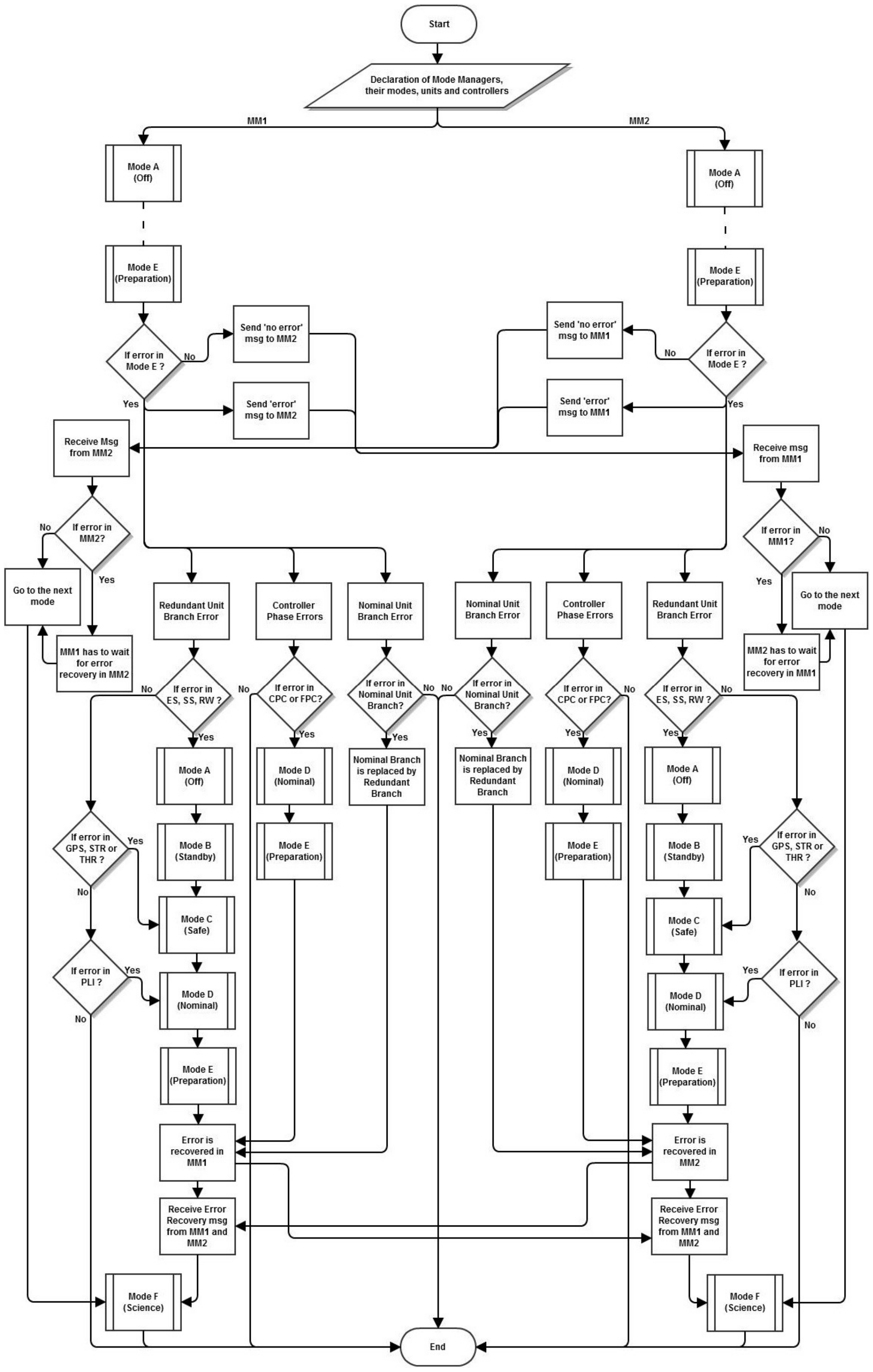
Start
Declaration of Mode Managers, their modes, units and controllers
MM1
MM2
Mode A (Off)
Mode E (Preparation)
If error in Mode E ?
No
Yes
Send 'no error' msg to MM2
Send 'error' msg to MM2
Send 'no error' msg to MM1
Send 'error' msg to MM1
Receive Msg from MM2
Receive msg from MM1
If error in MM2?
If error in MM1?
Go to the next mode
MM1 has to wait for error recovery in MM2
MM2 has to wait for error recovery in MM1
Redundant Unit Branch Error
Controller Phase Errors
Nominal Unit Branch Error
If error in ES, SS, RW ?
If error in CPC or FPC?
If error in Nominal Unit Branch?
Nominal Branch is replaced by Redundant Branch
Mode A (Off)
Mode B (Standby)
Mode C (Safe)
Mode D (Nominal)
Mode E (Preparation)
If error in GPS, STR or THR ?
If error in PLI ?
Error is recovered in MM1
Error is recovered in MM2
Receive Error Recovery msg from MM1 and MM2
Mode F (Science)
End

Figure 3: System flow chart for Mode E to Mode F

```
        /* The remaining part of the code, by calling the handshake
        protocol function on the basis of unit and controller error flag, is
        mentioned in Section IV.*/}
    else if((N_ES1!=off || N_SS1!=off || N_RW1!=on) && R_ES1==off
    && R_SS1==off && R_RW1==on && (N_ES2==off || N_SS2!=off
    || N_RW2!=on) && R_ES2==off && R_SS2==off && R_RW2==on
    ) {   u_MM1=1;c_MM1=0;
        u_MM2=1;c_MM2=0;
        /* The remaining part of the code, by calling the handshake
        protocol function on the basis of unit and controller error flag, is
        mentioned in Section IV.*/}
    else{
        /* The associated code describes that no transition takes place.
        */ }        }
else    cout<<" Program is terminated.";}
```

Our verification efforts are focused on checking correctness of mode syncornization and verification of the proposed collaboration scheme. To obtain quantitative measures of the performance of the discussed protocol we would need to further refine our specification and to integrate model of hardware platform in the loop. We are planning to perform quantitative evaluation as a part of the future work.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we demonstrated how to model and verify distributed satellite systems with complex mode transition logic. Our approach is validated by a case study – design of a distributed Attitude and Orbit Control System.

The proposed system has been implemented in SystemC language. SystemC specification can be easily interfaced with various model checking techniques to perform formal verification. The work presented in this paper extends our previous work done on modeling centralized mode-rich system. In the current approach, we have put the main focus on mode synchronization aspect and demonstrated how to achieve mode consistency via handshaking protocol.

Our work complements research done on formal modeling of mode-rich satellite systems. The formal modeling proposed by Iliasov et al. [8,9] focused on proof-based verification of centralized AOCS. Formal modeling of the distributed architecture presented in our paper is a completely novel aspect.

As a future work, we are planning to investigate how to interface architectural modeling with our design approach.

REFERENCES

[1] "DEPLOY Work Package 3 - Software Requirements Document for Distributed System for Attitude and Orbit Control for a Single Spacecraft", Space Systems Finland, Ltd., June 2011[retrieved: November, 2011].

[2] "DEPLOY Work Package 3 - Attitude and Orbit Control System Software Requirements Document", Space Systems Finland, Ltd., December 2010 [retrieved: January, 2012].

[3] J. Kashif, and E. Troubitsyna, "Designing a Fault-Tolerant Satellite System in SystemC", ICONS 2012, The Seventh International Conference on Systems, XPS Press, pp. 49-54, March 2012.

[4] M. Heimdahl, and N. Leveson, "Completeness and Consistency in Hierarchical State-Based Requirements", IEEE Transactions on Software Engineering, Vol.22, No. 6, pp. 363-377, June 1996.

[5] N. Leveson, L. D. Pinnel, S. D. Sandys, S. Koga, and J. D. Reese, "Analyzing Software Specifications for Mode Confusion Potential", Proceedings of Workshop on Human Error and System Development, C.W. Johnson, Editor, Glasgow, Scotland, pp. 132-146, March 1997.

[6] N. Blanc, D. Kroening, and N. Sharygina, "Scoot: A Tool for the Analysis of SystemC Models". TACAS'08/ETAPS'08 Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the Construction and Analysis of Systems, Springer-Verlag, Berlin, Heidelberg, pp. 467–470, 2008.

[7] L. Singh, and L. Drucker, "Advanced Verification Techniques: A SystemC Based Approach for Successful Tapeout", Kluwer Academic Publishers, Springer, 2004.

[8] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala, "Developing Mode-Rich Satellite Software by Refinement in Event B". In: Proc.of FMICS 2010, the 15th International Workshop on Formal Methods for Industrial Critical Systems, Lecture Notes for Computer Science, Springer, 2010.

[9] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, P. Väisänen, D. Ilic, and T. Latvala, "Verifying Mode Consistency for On-Board Satellite Software". In Proc. of SAFECOMP 2010, The 29th International Conference on Computer Safety, Reliability and Security, September 14-17, Vienna, Austria, Lecture Notes for Computer Science, Springer, September 2010.

[10] "DEPLOY deliverable D20 – Pilot Deployment in the Space Sector", Space Systems Finland, Ltd., January 2010 [retrieved: March, 2012].