Enabling Effective Requirements Compositional Verification for Complex Embedded Systems – Supplemental Material

1 Introduction

This is the supplementary document of paper "Enabling Effective Requirements Compositional Verification for Complex Embedded Systems". It provides 5 case study's original requirements and experiment details.

The rest of this document is organized as follows. Section 2 gives the sun search subsystem's original requirements. Section 3 outlines the procedural steps involved in identifying requirement dependencies within the case study experiment. Section 4 describes how to generate architectures of each cases. Section 5 presents an evaluation of comparison our work with existing research.

2 The Sun Search Subsystem's Original Requirements

The solar search subsystem is to collect the data from the gyros, the sun sensors and thrusters, and automatically control the satellite to rotate towards the sun according to the data collected and instructions from the ground through data management computer. The subsystem has three passive device (Gyro, Sun sensor, Thruster), an active device (Data Management Computer), a clock (32ms Interrupt Timer) and 20 original requirements. The 20 original requirements are as follows.

1. **Store Gyro Power-on Instruction:** When the system is initialized, the controller needs to store gyro power-on instructions.
2. **Store Sun Sensor Power-on Instruction:** When the system is initialized, the controller needs to store sun sensor power-on instructions.
3. **Store Thruster Power-on Instruction:** When the system is initialized, the controller needs to store thruster power-on instructions.
4. **32ms Interrupt Timer Initialization:** When the system is initialized, the controller needs to send start instruction to 32ms interrupt timer, the timer enter start state after receiving the instruction.
5. **Gyro Data Acquisition:** Clock drives controller to acquire pulse-count, gyro power state from gyro. The controller gets angular velocity analog from pulse-count and stores angular velocity analog, gyro power state.
6. **Sun Sensor Data Acquisition:** Clock drives controller to acquire sun sensor power state, sun visible sign and sun measurement angle from sun sensor. The controller needs to store the collected data.

7. **Thruster Data Acquisition:** Clock drives controller to acquire thruster power state, jet interval from thruster. The controller needs to store the collected data.
8. **Gyro Control Output:** Clock drives controller to transform gyro power-on instruction/gyro power-off instruction into gyro power-on signal/gyro power-off signal. Then the controller needs to send signal to gyro.
9. **Sun Sensor Control Output:** Clock drives controller to transform sun sensor power-on instruction/sun sensor power-off instruction into sun sensor power-on signal/sun sensor power-off signal. Then the controller needs to send signal to sun sensor.
10. **Thruster Power Control Output:** Clock drives controller to transform thruster power-on instruction/thruster power-off instruction into thruster power-on signal/thruster power-off signal. Then the controller needs to send signal to thruster.
11. **Thruster Triaxial Control:** Clock drives controller to transform triaxial control instruction into triaxial control signal. Then the controller needs to send signal to thruster. The thruster enters jet state.
12. **Store Gyro Power-off Instruction:** Clock drives controller to judge gyro fault by using gyro communication error time. If gyro communication error time is greater than 5 seconds, controller turns off the gyro.
13. **Store Sun Sensor Power-off Instruction:** Clock drives controller to judge sun sensor fault by using sun visible sign. If the sun visible sign is still invisible after two consecutive pitch search and scroll search, controller turns off the sun sensor.
14. **Store Thruster Power-off Instruction:** Clock drives controller to judge thruster fault by using jet interval. If the jet interval is less than one second in five consecutive seconds, controller turns off the thruster.
15. **Attitude Determination:** Clock drives controller to calculate triaxial attitude angle and triaxial angular velocity using angular velocity analog, sun visible sign, sun measurement angle, current mode.
16. **Mode Switching Management:** Clock drives controller to calculate next cycle mode using triaxial attitude angle, triaxial angular velocity, sun visible sign, current mode.
17. **Control Computing:** Clock drives controller to calculate triaxial control quantity using triaxial attitude angle, triaxial angular velocity, target angle, target angular velocity.
18. **Telecontrol Instruction Processing:** The controller needs to receive telecontrol instruction from data management computer, and store next cycle mode parsed from telecontrol instruction.
19. **Telemetry Processing:** Clock drives controller to pack current mode, triaxial attitude angle, triaxial angular velocity to generate telemetry data. The controller sends telemetry data to data management computer.

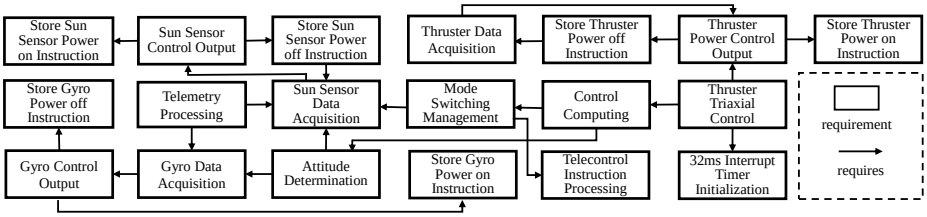


Fig. 1. The requirements dependency diagram of the SSCS.

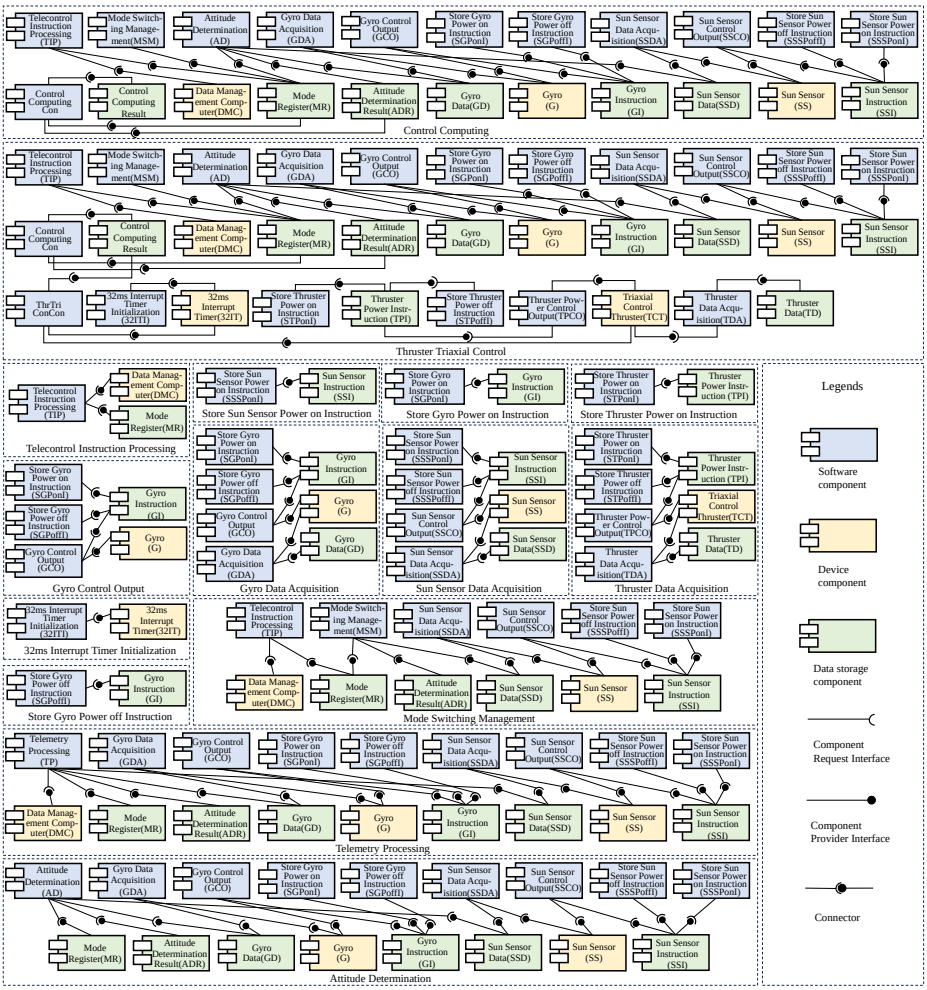


Fig. 2. The component diagram of the SSCS.

3 Identify dependencies

4 Generate architecture

5 Verify results and Evaluation

The purpose of evaluation is to answer the following question: Can this approach reduce the state space required for the complex embedded system requirements verification, and improve the efficiency of verification? We are contemplating the use of comparative experiments to juxtapose the composition verification outcomes of our approach which considering dependencies against the all-model verification results.

Experiment Preparation: In this experiment, we have selected five different case studies, namely the light control system (LCS), smart home control subsystem (SCS), rail transit vehicle control subsystem (VOBC), airborne reconnaissance control subsystem (RCS), and the aerospace sun search control subsystem (SSCS), as shown in Table 1. Among them, the LCS and SCS are toy examples sourced from academic literature. Conversely, the SCS, VOBC, RCS, and SSCS are derived from real-world scenarios. We choose different requirements numbers to represent different scales of systems. In addition, we construct four compact device knowledge libraries for the four domains of five cases by performing TA modeling on the devices involved. The software environment used for the experiment is UPPAAL-4.1.24, and the hardware environment is a Windows 10 system with 32GB of RAM and 1TB SSD.

Table 1. Details of cases chosen, where DE stands for dependency error, while SE means single requirement related error.

Case	LCS	SCS	VOBC	RCS	SSCS
Source	[1]	[2]	real-world	real-world	real-world
#Device	5	15	7	12	28
#Requirements	6	11	4	19	19
Errors embedded	DE	SE	–	SE and DE	DE

Experiment Procedure:

Firstly, we use our approach to verify the five cases. By inputting the sequence diagrams of each case, supported with the corresponding device model library, we identify the device and data related dependencies, according to which we generate the architecture for each verification subsystem and create an executable model for subsystem verification. Lastly, we use an application independent property, e.g., no deadlock for verification. Deadlocks, representing unexpected or premature system halts, are generally undesirable in system models. we carry out no deadlock verification for the verification subsystems in each case.

Secondly, For the comparative experiment addressing this issue, We use the all-model verification approach, and combine all the software and device components in the subsystems of the verification for no deadlock verification. We

record the verification results, the state space and runtime of the verification system.

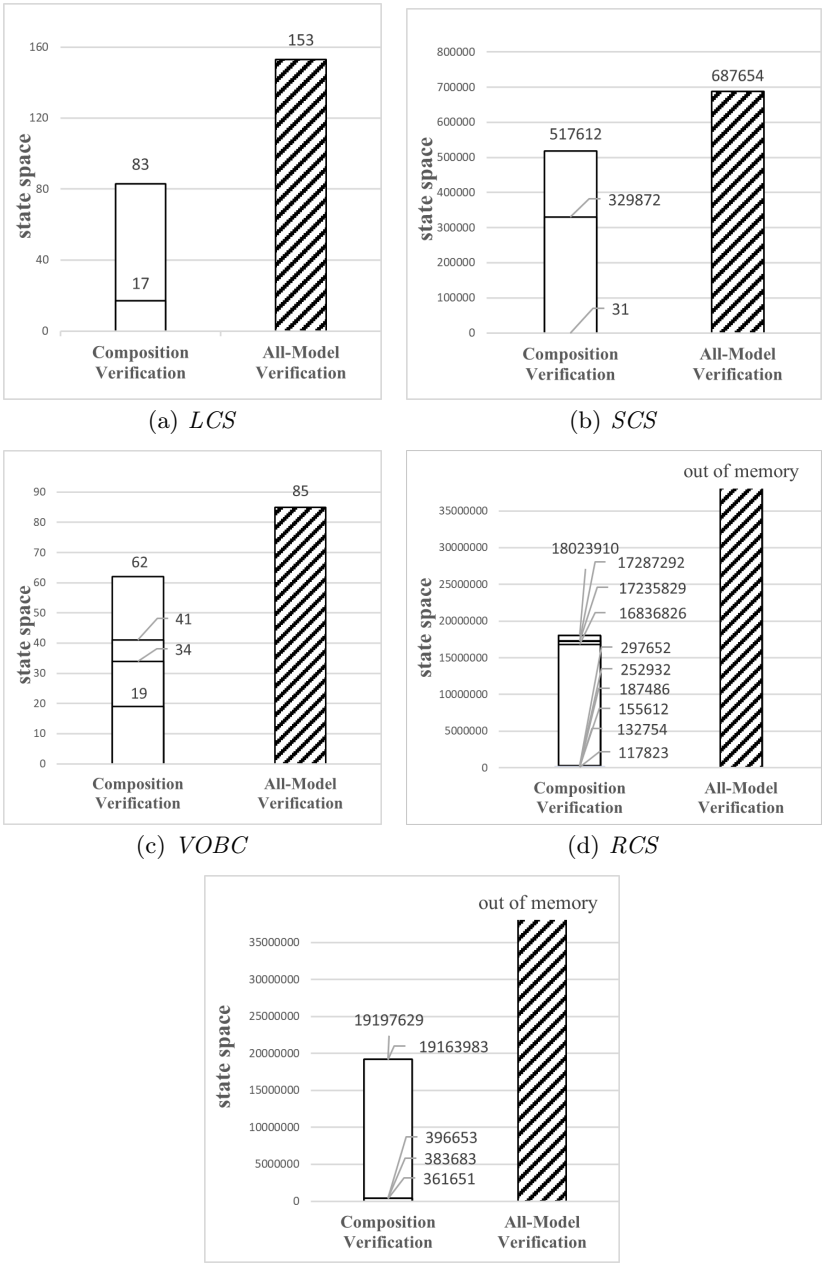
Finally, we organized all the related verification data. For our approach, we record verification results, states traversed and verification time for each verification system of each case. For the all-model verification, we record its verification result, the state space and verification time for each case. Considering that factors such as the operating system may impact the experimental results, we conducted ten times for each experiment. We took the average of these values. The final results are shown in Table 2, Fig. 3, and Fig. 4.

Table 2. Evaluation results, where ✓(×) means satisfied (dissatisfied), ver. sub. stands for verification subsystem, ver. res. means verification results, syn. res. means the synthesized result, and max. (min.) sta. exp. means maximum (minimum) states explored.

Case	**Our approach**						**All-model verification**	
	requirements dependencies	#ver. sub.	ver. res.	syn. res.	#max sta. exp.	#min sta. exp.	ver. res.	#sta. exp.
LCS	3 data related 2 device related	6	5 ✓ 1 ×	×	79	17	×	143
SCS	5 data related	11	9 ✓ 2 ×	×	517955	31	×	697901
VOBC	2 data related	4	4 ✓	✓	54	19	✓	78
RCS	11 data related 6 device related	19	16 ✓ 3 ×	×	18025481	117823	——	——
SSCS	16 data related 5 device related	15	10 ✓ 5 ×	×	19199651	361651	——	——

Result analysis: From Table 2, it can be seen that our compositional verification approach performs well. It can get the same results (including dependency errors) as the all-model verification while reducing the state space of verification subsystems when used for small scale applications. For larger cases like RCS and SSCS, the all-model verification was not able to give a result due to limited capability of UPPAAL, while our approach is able to given a result in limited time. This shows that our approach has the ability to deal with complex embedded systems. In fact, the number of states explored by these two complex systems during all-model verification exceeds one billion, while through our approach, the number of states explored by its subsystems is at most in the tens of millions.

From the perspective of verifiable systems, i.e., LCS, SCS, and VOBC, our approach has already reduced the explored state space by at least 25.78% shown in Fig 3. However, this does not mean that the sum of the states explored by each verification subsystem is necessarily less than the states of the all-model verification system. For example, in the SCS, the sum of the states explored by

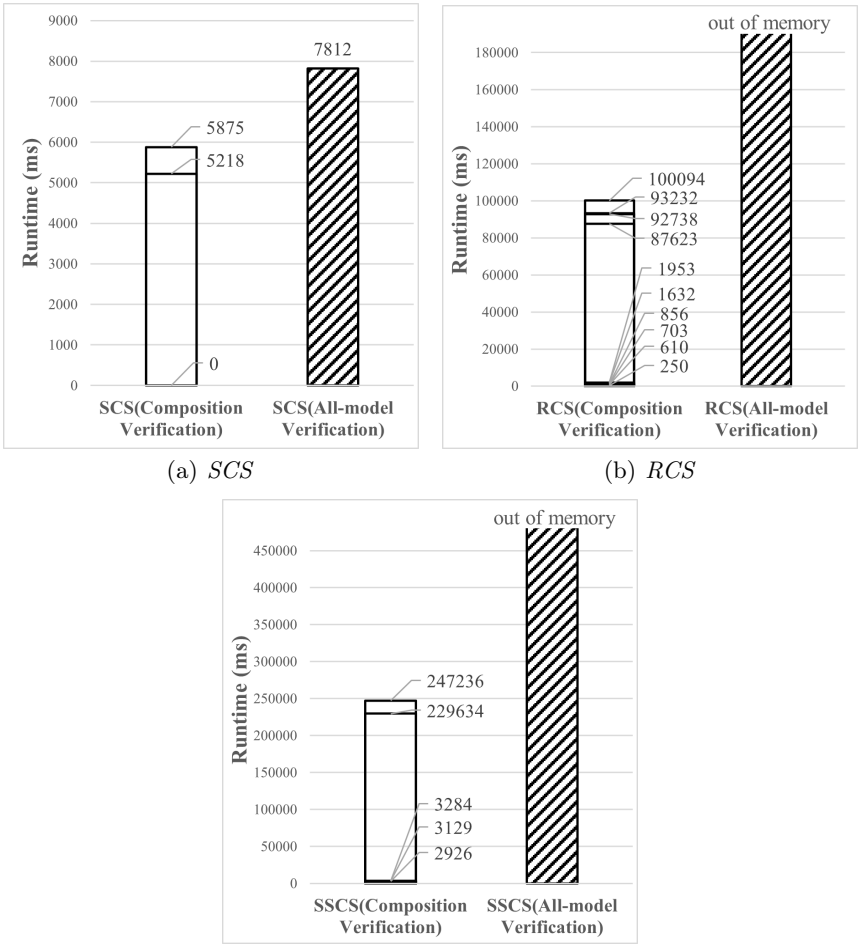


(a) *LCS* (b) *SCS*

(c) *VOBC* (d) *RCS*

(e) *SSCS*

Fig. 3. States space verification results for 5 case studies.



(a) *SCS* (b) *RCS*

(c) *SSCS*

Fig. 4. States space verification results for 5 case studies.

the two sub-verification systems has already exceeded the all-model verification state. This indicates that our approach does not have practical application value on small-scale systems. However, we can use them to demonstrate the reduction of state space in verification subsystems.

Regarding verification efficiency, Table 2 shows that for simpler cases like the LCS and VOBC, the verification state space is under 100, taking almost no time during verification. However, for more complex systems like SCS, the state space can reach hundreds of thousands. Comparing the maximum time for each subsystem verification with the all-model verification shows a minimum 24.80% reduction in time consumption shown in Fig 3. Note, this reduction refers to each subsystem versus the all-model verification, not the total verification time of all subsystems, which might exceed that of the all-model system. Thus, for systems that can perform all-model verification at this scale, our approach may not qbe practical. For systems like the RCS and SSCS, which are even more complex, the verification space size has reached tens of millions. The all-model verification method fails due to memory overflow, while our approach can complete verification, with a maximum time reaching 247.136s shown in Fig 4. This shows that our approach can significantly improve efficiency in handling complex system requirement verification.

References

1. M, J.: Problem frames: analysing and structuring software development problems. Addison-Wesley (2001)
2. Pohl, K., Sikora, E.: Overview of the example domain: Home automation. In: Software Product Line Engineering - Foundations, Principles, and Techniques, pp. 39–52. Springer (2005)