Brief Industry Paper: Modeling and Verification of Descent Guidance Control of Mars Lander

Bohua Zhan
State Key Lab. of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing, China
bzhan@ios.ac.cn

Bin Gu
Beijing Institute of Control Engineering
Beijing, China
gubinbj@sina.com

Xiong Xu, Xiangyu Jin, Shuling Wang, Bai Xue
State Key Lab. of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing, China
{xux,jinxy,wangsl,xuebai}@ios.ac.cn

Xiaofeng Li, Yao Chen
Beijing Institute of Control Engineering
Beijing, China
li_x_feng@126.com, yushengluck@163.com

Mengfei Yang
China Academy of Space Technology
Beijing, China
yangmf@bice.org.cn

Naijun Zhan
State Key Lab. of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing, China
znj@ios.ac.cn

***Abstract*—We give an introduction to the MARS toolchain for formal modeling and verification of hybrid systems. It consists of translators from Simulink/Stateflow models to Hybrid Communicating Sequential Processes (HCSP), and tools for simulation, code generation, and deductive verification of an HCSP model. We apply the toolchain to model the descent guidance control phase of the recently launched Tianwen I mars lander, and verify that it correctly controls the velocity of the lander.**

***Index Terms*—HCSP, Simulink, simulation, verification, code generation**

I. Introduction

Embedded systems play a key role in industry and in our daily lives. They help control everything from aircraft and spacecraft, to high-speed trains, to nuclear power plants and medical devices. Many of these applications are safety-critical, in the sense that errors can lead to loss of human life or failure of a mission. Hence, it is essential to guarantee the reliability of these embedded systems.

A common approach to improve reliability of a system is to follow a model-driven design process. This means first constructing an abstract model of the system to be implemented, extensively simulating and verifying the model on a computer in order to identify and correct design problems early, and finally implementing the model as a concrete system, perhaps with the help of automatic code generation. There are many tools available for model-driven design, including Matlab/Simulink and Scade. They have already been applied successfully in the automobile, aerospace, and other industries.

Both Matlab/Simulink and Scade are graphical modeling languages, meaning engineers can draw diagrams on the computer that represent the model. This provides a very intuitive environment for the engineer to work in. However, there can be ambiguity in the exact meaning of these models. Moreover, simulation can only check correctness of the model on the given inputs. More advanced methods, such as reachability computation or deductive verification, are needed to verify that a model is correct on all valid inputs.

MARS (Modeling, Analyzing and veRifying hybrid Systems) is a toolchain for modeling and verification of embedded systems. The language lying at the center of the toolchain, Hybrid CSP, is designed to model hybrid systems containing both continuous and discrete behavior. It gives formal meanings to a subset of Simulink by translating the model to Hybrid CSP. Then, it provides tools for simulation, code generation, and deductive verification of an HCSP model. The MARS toolchain has been used successfully in industrial applications [1], [2].

In this paper, we apply the MARS toolchain to the modeling and verification of part of the control program for the recently launched Tianwen I mars rover. We concern ourselves with the descent guidance control phase, which is used to maintain the downward velocity of the rover during the final phase of descent. We model the control program using Simulink, and translate the model to HCSP. We analyze the model by simulation, as well as verify that it maintains the given velocity bounds using deductive methods. Finally, we generate SystemC code from the model.

II. The Mars Toolchain

In this section, we give an introduction to the different parts of the MARS toolchain: the Hybrid CSP language, translation from Simulink, as well as its simulation, code generation, and deductive verification tools.

A. *Hybrid CSP*

Hybrid CSP is a language for modeling hybrid systems: systems with both continuous and discrete behavior. It can also describe several processes running in parallel, and synchronized communication between processes. The syntax of the language is as follows:

$$\begin{array}{lll} P & ::= & \text{skip} \mid x := e \mid ch?x \mid ch!e \mid P;Q \mid \langle \dot{x} = e \& B \rangle \mid \\ & & \text{wait } d \mid \text{if } B \text{ then } P_1 \text{ else } P_2 \mid P \sqcup Q \mid P^* \mid \\ & & \talloblong_{i\in I}(io_i \to Q_i) \mid \langle \dot{x} = e \& B \rangle \trianglerighteq \talloblong_{i\in I}(io_i \to Q_i) \\ S & ::= & P \mid S \|_{cs} S \end{array}$$

The intuitive meaning of some of the constructs are as follows. See [3] for more details.

- wait d statement will keep idle for d time units keeping variables unchanged.
- Input $ch?x$ receives a value along the channel ch and assigns it to variable x. Output $ch!e$ sends the value of e along ch. Input and output must be *synchronized*, meaning one must wait for the other to be ready before proceeding.
- Repetition P^* executes P for a nondeterministic finite number of times.
- External choice $\talloblong_{i\in I}(io_i \to Q_i)$ waits for the communication events io_i, and carries out the first event that can occur, followed by the corresponding Q_i.
- $\langle \dot{s} = e \& B \rangle$ is the continuous evolution construct, where s is a vector of variables and e is a vector of expressions. The state will evolve continuously according to the differential equation $\dot{s} = e$ as long as the *domain* B holds, and terminates when B becomes false.
- Communication interrupt $\langle \dot{s} = e \& B \rangle \trianglerighteq \talloblong_{i\in I}(io_i \to Q_i)$ behaves like $\langle \dot{s} = e \& B \rangle$, except it is preempted as soon as one of the communication events io_i takes place, and then is followed by the corresponding Q_i.
- $S_1 \|_{cs} S_2$ describes two processes S_1 and S_2 running concurrently, with all communications along the channels cs synchronized. We assume S_1 and S_2 do not have shared memory, and so can transfer information only through communication channels.

B. *Translation from Simulink to HCSP*

Matlab/Simulink provides a graphical language for describing system models. A Simulink diagram consists of a set of blocks. Each block has a set of input signals and computes an output signal. A block may be *discrete*, meaning it computes the output signal from the input signals periodically. Alternatively, a block may be *continuous*, meaning both input and output signals are continuously varying functions of time. Combinations of continuous blocks can describe differential equations. Blocks can also be organized into hierarchies of subsystems.

MARS provides a tool for translating Simulink (as well as Stateflow) diagrams to HCSP. This allows engineers to construct models to be used in MARS in a graphical environment. We describe the translation procedure briefly as follows. First, all subsystems in the diagram are unfolded. Then, the diagram is divided into parts consisting of continuous and discrete blocks, respectively. A group of discrete blocks is translated to an HCSP process consisting mostly of assignments, while a group of continuous blocks is translated to a process containing continuous evolution. Data-exchanges between the continuous and discrete blocks are translated to communications among processes. For more details about the translation algorithm and proof of its correctness, see [4] and the book [3].

More concretely, the translator is implemented as a Python package ss2hcsp. It takes the Simulink model in XML format as input, and outputs the corresponding HCSP model. This can then be used by the downstream tools in the MARS toolchain.

C. *Simulation of HCSP Model*

A simulator for HCSP is implemented in Python, together with a web interface for viewing simulation results. The simulator can be used to check that the translation from Simulink to HCSP does preserve the behavior of the model. It can also be used for developing HCSP models directly.

ODE solving in the simulator is done using Python's scipy package. The package is able to accurately calculate the time at which the boundary of the domain is reached using a root-finding algorithm. The web interface shows the code of the HCSP model alongside a trace of execution, and a plot of the variables in the process against time. This allows us to not only view the results of running a model, but also help debug the model if it does not execute as expected.

D. *Code generation to SystemC*

The translator HCSP2SC takes an HCSP program as input, and generates the corresponding SystemC code. SystemC is a system-level modeling language for hardware and software co-design. We choose SystemC as the target language because it is widely adopted in the design of embedded systems. Moreover, its syntax and semantics are very close to that of HCSP, in particular with support for communication and synchronization, making the translation more straightforward.

For example, consider the input operation $ch?x$, the corresponding SystemC code is shown as follows. It first initializes the signal `ch_r` to `1`, showing the process is ready to read (line 1); it waits for the output to be ready and finishes writing (lines 2–3); gets the latest value from the channel and assigns it to the variable `x` (line 4); informs that it has finished reading (line 5); and finally resets `ch_r` to `0` (line 6). The output operation $ch!e$ can be translated in a similar way.

```
1  ch_r = 1;
2  if (!ch_w) wait(ch_w.posedge_event());
3  wait(ch_w_done);
4  x = ch.read();
5  ch_r_done.notify();
6  ch_r = 0;
```

For the continuous behaviour of an HCSP process, a discretization is performed. The notion of approximate bisimulation is used as a criterion to check the consistency between the original HCSP process and its discretization. The proof of approximate bismulation between the HCSP process and generated code (under certain assumptions) is given in [5].

E. Deductive Verification

Simulation is only able to check the behavior of the model for certain initial conditions and inputs. For greater confidence in the correctness of the model, we can use methods based on logic to verify that the model behaves correctly on all valid initial conditions and inputs.

Deductive verification of Hybrid CSP is based on Hybrid Hoare Logic (HHL). For previous work on Hybrid Hoare Logic, see [6] for the theoretical formulation and [7] for the implementation in Isabelle/HOL. In Hybrid Hoare Logic, the correctness of a program is described by a *Hoare triple*, of the form $\{P\}\ c\ \{Q\}$, where c is the program, P is the precondition, and Q is the postcondition. It means that if property P holds in the initial state, then after running c, the final state satisfy property Q. The precondition P is used to specify initial conditions, while Q is used to specify the properties the program is expected to have.

III. The Simulink Model of the Mars Lander

The descent guidance control phase is the last phase of the descent of the mars rover onto the surface. The main goal of this phase is to maintain a stable downward velocity of the rover. For this demonstration, we consider a simplified version of the model. The values of key parameters are omitted or given in ranges in this paper.

The ODE describing continuous evolution is given by:

$$\begin{cases} \dot{s} = v \\ \dot{v} = \frac{F_c}{m} - g_M \\ \dot{m} = \frac{-F_c}{Isp} \end{cases}$$

where v is the downward velocity and m the mass of the lander. g_M is the acceleration due to gravity on the martian surface and F_c is the thrust imposed on the lander (a constant in each sampling period). Isp is the *specific impulse*, measuring the rate at which mass is lost for different settings of applied force. It is modeled as a piecewise-constant function of F_c consisting of five constant segments, in the form

$$Isp = \begin{cases} Isp_0 & \text{for } F_c > F_{c1} \\ Isp_1 & \text{for } F_{c1} \geq F_c > F_{c2} \\ Isp_2 & \text{for } F_{c2} \geq F_c > F_{c3} \\ Isp_3 & \text{for } F_{c3} \geq F_c > F_{c4} \\ Isp_4 & \text{for } F_{c4} \geq F_c \end{cases}$$

The thrust is updated every period of P_{ctrl} seconds, with the control law given by:

$$F_c' = m \cdot (g_M - c_2(v - v_{slw}) - c_3(\frac{F_c}{m} - g_M))$$

where v_{slw} is the target velocity that should be maintained throughout the descent process (some fixed value in the range 1–5m/s). Moreover F_c is limited to the range $[F_{min}, F_{max}]$.

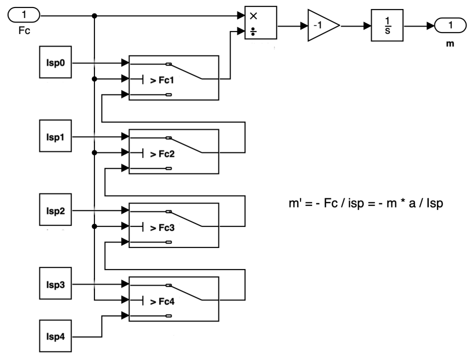

Fig. 1. The continuous evolution of mass m

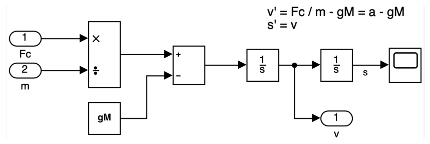

Fig. 2. The continuous evolution of velocity v and height s

We first create models for descent guidance control using Simulink diagrams. They are shown in the following figures. Fig. 1 shows the model for the equation $\dot{m} = \frac{ma}{Isp}$, including switch blocks for the computation of Isp. Fig. 2 shows the model for the equations $\dot{s} = v$ and $\dot{v} = \frac{F_c}{m} - g_M$. Finally, Fig. 3 shows the computation of control law.

IV. Simulation and Code Generation

A. Translation to HCSP

In this section, we show the result of translation from Simulink to HCSP. In our case, the models in Fig. 1 and Fig. 2 together form a group of continuous blocks describing the ODE for s, v and m. This is translated into the following HCSP process:

$$\begin{array}{l} Plant ::= \\ s := s_{init}; v := v_{init}; m := m_{init}; ch_m!m; ch_v!v; ch_{Fc1}?Fc; \\ \left(\begin{array}{l} \text{if} \quad Fc \leq F_{c4} \quad \text{then} \\ \quad \langle \dot{s} = v, \dot{v} = \frac{Fc}{m} - g_M, \dot{m} = \frac{Fc}{-Isp_4} \& Fc \leq F_{c4} \rangle \\ \quad \trianglerighteq [\!]\ (ch_m!m \rightarrow (ch_v!v; ch_{Fc1}?Fc)) \\ \text{elif} \quad F_{c4} < Fc \leq F_{c3} \quad \text{then} \quad \cdots \\ \text{elif} \quad F_{c3} < Fc \leq F_{c2} \quad \text{then} \quad \cdots \\ \text{elif} \quad F_{c2} < Fc \leq F_{c1} \quad \text{then} \quad \cdots \\ \text{else} \quad \cdots \quad \text{endif} \end{array} \right)^* \end{array}$$

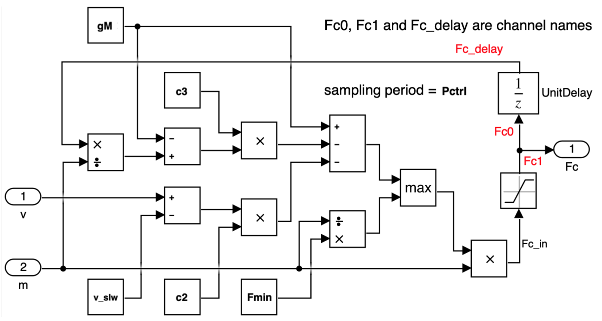

Fig. 3. The Simulink diagram of the discrete control

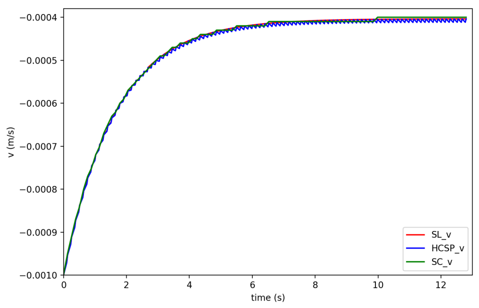

Fig. 4. The simulation results of the original Simulink model (red), the corresponding HCSP process (blue) and the generated SystemC code (green). Velocity axis gives offset to v_{slw}.

Fig. 3 shows the discrete part of the system, with a sample period of P_{ctrl} seconds. Of all discrete blocks, the unit delay UnitDelay needs special treatment, because it records the value received in the previous cycle, i.e., it maintains state while the other blocks do not. Therefore, it is translated into a separate HCSP process:

$$\begin{aligned} \mathit{UnitDelay} \quad ::= \quad & Fc := Fc_{init};\, ch_{Fc\_delay}!Fc; \\ & (ch_{Fc0}?Fc;\, \text{wait } P_{ctrl};\, ch_{Fc\_delay}!Fc)^* \end{aligned}$$

The remaining discrete blocks are translated directly into the following HCSP process:

$$\begin{aligned} & \mathit{Control} ::= \\ & \left(\begin{array}{l} ch_m?m;\, ch_{Fc\_delay}?Fc_{delay};\, ch_v?v; \\ Fc_{in} := m \cdot \max(\frac{F_{min}}{m}, g_M - c_2 \cdot (v - v_{slw}) \\ \qquad\qquad\qquad -c_3 \cdot (\frac{Fc_{delay}}{m} - g_M)); \\ \text{if } Fc_{in} < F_{min} \text{ then } Fc := F_{min} \text{ elif } Fc_{in} > F_{max} \\ \text{then } Fc := F_{max} \text{ else } Fc := Fc_{in} \text{ endif}; \\ ch_{Fc0}!Fc;\, ch_{Fc1}!Fc;\, \text{wait } P_{ctrl} \end{array} \right)^* \end{aligned}$$

In summary, the entire system is translated into the parallel composition of the above three processes:

$$\mathit{Lander} \quad ::= \quad \mathit{Plant} \parallel \mathit{UnitDelay} \parallel \mathit{Control}$$

B. Results of simulation

Fig. 4 shows the simulation results of the model. It shows how the velocity v changes over time. Results from the Simulink model is shown in red and the results from the translated HCSP process is shown in blue. Simulation also shows that the height s and mass m steadily decreases as expected (we omit the figures due to lack of space). The results show that the behavior of the model before and after translation is broadly consistent with each other.

C. From HCSP to SystemC

We use tool HCSP2SC to automatically generate SystemC code from the HCSP model. In Fig. 4, the green line shows the result of running the generated code. It can be seen that the results practically coincide with the simulation results from the Simulink and HCSP models.

V. Deductive Verification

The goal of deductive verification is to prove that if the initial conditions stay in a reasonable range, then the velocity can be maintained around v_{slw}. More specifically, we verify that the velocity satisfies $|v - v_{slw}| < 0.05$.

A key step during the verification is finding a suitable invariant. Let a be the acceleration $\frac{F_c}{m} - g_M$. We aim to find a function $F(v, a, t)$ of velocity, acceleration, and time within a period, such that if $F(v, a, t) < 0$ initially, then it will stay the case throughout the evolution of the system, including both continuous evolution and the control law update. This reduces to the invariant satisfying the following three conditions.

1) It is preserved by continuous evolution, which can be guaranteed by the condition that whenever $F(v, a, t) = 0$, then $\frac{d}{dt}F(v, a, t) < 0$.
2) It is preserved by the update of F_c at the end of each period. That is, if $F(v, a, P_{ctrl}) < 0$, then $F(v, a', 0) < 0$, where a' corresponds to the new value of F_c.
3) It implies the postcondition. That is, if $F(v, a, t) < 0$, then v lies in the interval $[v_{slw} - 0.05, v_{slw} + 0.05]$.

The invariant is found using standard techniques [8], using Yalmip in Matlab and checked in Mathematica. Once the invariant is found, we apply hybrid Hoare logic to verify a Hoare triple expressing that velocity stays in the range $[v_{slw} - 0.05, v_{slw} + 0.05]$. The proof is checked in Isabelle/HOL.

VI. Conclusion

In this paper, we presented the Mars toolchain for modeling and verifying embedded systems, and a case study on the descent guidance control phase of the Tianwen I mars rover. Compared to previous studies [1], we demonstrate two recently implemented features of the toolchain: a simulator for HCSP and automatic code generation to SystemC, as well as recent updates to Hybrid Hoare Logic and the translator from Simulink diagrams to HCSP.

Acknowledgment

This work has been partially funded by NSFC under grant No. 61625206, 61732001 and 61872341, and by the CAS Pioneer Hundred Talents Program under grant No. Y9RC585036.

References

[1] H. Zhao, M. Yang, N. Zhan, B. Gu, L. Zou, and Y. Chen, "Formal verification of a descent guidance control program of a lunar lander," in *FM 2014: Formal Methods, Singapore, May 12-16*, 2014, pp. 733–748.

[2] E. Ahmad, Y. Dong, B. R. Larson, J. Lü, T. Tang, and N. Zhan, "Behavior modeling and verification of movement authority scenario of Chinese train control system using AADL," *Sci. China Inf. Sci.*, vol. 58, no. 11, pp. 1–20, 2015.

[3] N. Zhan, S. Wang, and H. Zhao, *Formal Verification of Simulink/Stateflow Diagrams–A Deductive Approach*. Springer, 2017.

[4] L. Zou, N. Zhan, S. Wang, M. Fränzle, and S. Qin, "Verifying Simulink diagrams via a hybrid Hoare logic prover," in *EMSOFT 2013, Montreal, QC, Canada, September 29 - Oct. 4*, 2013, pp. 9:1–9:10.

[5] G. Yan, L. Jiao, S. Wang, L. Wang, and N. Zhan, "Automatically generating SystemC code from HCSP formal models," *ACM Trans. on Soft. Eng. and Methodology*, vol. 29, no. 1, pp. 4:1–4:39, January 2020.

[6] D. P. Guelev, S. Wang, N. Zhan, and C. Zhou, "Super-dense computation in verification of hybrid CSP processes," in *FACS 2013, Nanchang, China, October 27-29, Revised Selected Papers*, 2013, pp. 13–22.

[7] S. Wang, N. Zhan, and L. Zou, "An improved HHL prover: An interactive theorem prover for hybrid systems," in *ICFEM 2015, Paris, France, November 3-5*, 2015, pp. 382–399.

[8] H. Kong, F. He, X. Song, W. N. N. Hung, and M. Gu, "Exponential-condition-based barrier certificate generation for safety verification of hybrid systems," in *CAV 2013, St. Petersburg, Russia, July 13-19*, 2013, pp. 242–257.