Electronic Communications of the EASST
Volume X (2013)

Proceedings of the Automated Verification of Critical Systems (AVoCS 2013)

Verifying sequence diagrams using the process algebra CSP

Tomohiro KAIZU , Yoshinao ISOBE , Masato SUZUKI ,

3 pages

Guest Editors: Steve Schneider, Helen Treharne
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Verifying sequence diagrams using the process algebra CSP

Tomohiro KAIZU [1], Yoshinao ISOBE [2], Masato SUZUKI [3],

[1] tkaizu@jaist.ac.jp, Japan Advanced Institute of Science and Technology
[2] y-isobe@aist.go.jp, National Institute of Advanced Industrial Science and Technology
[3] suzuki@jaist.ac.jp, Japan Advanced Institute of Science and Technology

Abstract: We develop a verification tool for sequence diagrams named SD2CSP. It converts sequence diagrams to processes in CSP, so that existing model checking tool can verify them. We implemented the tool and conducted experiments with real world case studies.

Keywords: Sequence Diagram, Process Algebra, CSP, Process Synthesis, Verification

Introduction UML diagrams are often used for designing software components. UML is a standardized modeling language developed by OMG. Especially in upstream development, UML sequence diagrams are frequently used to understand and verify the behavior of components.

However, the UML specification is complicated and flexible. Therefore, it is difficult to verify UML diagrams automatically. Users have relied on manual review to find mistakes such as inconsistencies and insufficient refinements between sequence diagrams. If such mistakes are found in a late development stage, it may take a lot of time and cost to correct them.

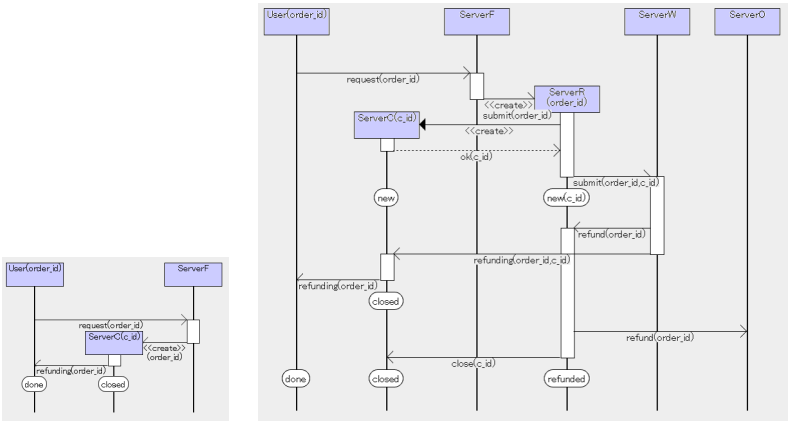
In [KIS13], we defined a subset of sequence diagrams with formal semantics and proposed a method to verify correctness of the sequence diagrams. In our method, sequence diagrams are converted to processes expressed in the process algebra CSP (Communicating Sequential Processes), and then the processes can be verified by model checking tools for CSP such as PAT and FDR2. The most important idea is to use internal choices and external choices for message sending and receiving, respectively. Using internal and external choices, our tool can generate a nondeterministic model, which is useful to verify early stage designs. In [KIS13], We implemented the conversion algorithm in a prototype tool named SD2CSP for demonstrating the effectiveness of the approach.

We develop a new version of SD2CSP for verifying sequence diagrams based on the approach presented in [KIS13]. The new SD2CSP supports following features for verifying real world systems.

- In sequence diagrams, objects can be dynamically created or destructed. The object ids can be passed by message parameters.
- Visualizing event traces on original sequence diagrams.

The tool is implemented with Dart language and compiled to JavaScript. Any browser which supports HTML5 and CSS3 can execute SD2CSP. Also, the editor and sample sequence diagrams can be found from the following page:

- http://dr.asukaze.net/sd2csp/online/



(a) (b)

Figure 1: sequence diagrams of Google Play store

Case Study We conducted a case study of SD2CSP using a real system in Google. The target model consists of 12 sequence diagrams: 4 sequence diagrams in abstract design and 8 sequence diagrams in concrete design. It represents servers in customer support system in Google Play store. We successfully verified it by SD2CSP and PAT tool version 3.2.1 that the concrete model is behaviorally equal to the abstract model by hiding internal communications.

Figure 1a is one of the abstract sequence diagrams. Only *User* and servers directly talking with *User* are defined in the abstract model. This diagram represents how this system works from the user's point of view. Figure 1b is one of the concrete sequence diagrams. More backend servers and messages are defined in the concrete model.

With deadlock checking, SD2CSP can verify this system will not accidentally stop. In SD2CSP, channels are assumed to be synchronous, therefore message pool is not generated automatically between processes. For example, if every process waits for sending messages, it is detected as a deadlock. When a deadlock is detected, the developer should write more sequence diagrams to specify what happens after the deadlock situation, or fix incorrect behavior so as not to reach the deadlock situation.

For refinement checking, we verified following assertion.

$$(AbstractModel \setminus X) \quad =_F \quad (ConcreteModel \setminus X)$$

where X is the set of events which the two models do not share. Note this is not an usual refinement checking in CSP. In the sequence diagram refinement process, new objects and messages are added to the model. It is not equivalent to the CSP refinement process. Instead, by hiding the newly added messages, it is expected that the abstract model and the concrete model are behaviorally equal because the user facing behavior is common in the two models.

Table 1 shows the result of the verification. In abstract level, 4 sequence diagrams with 5 objects are written. SD2CSP and PAT verified the model is deadlock-free in 0.026 seconds. In concrete level, 8 sequence diagrams with 10 objects are written. SD2CSP and PAT verified the model is deadlock-free in 0.337 seconds. With these 2 models, SD2CSP and PAT verified the

Table 1: result of the case study

	abstract model deadlock-free	concrete model deadlock-free	refinement checking	erroneous model deadlock-free
used time	0.026s	0.337s	0.575s	0.006s

abstract model and the concrete model are behaviorally equal from users' point of view using CSP refinements and hiding. It takes 0.575 seconds.

Furthermore, to check the ability for finding errors, we changed the model to intentionally introduce an error, and then applied SD2CSP to the erroneous model. SD2CSP successfully detected the bug in 0.006 seconds and the counter examples analysis tool supported us to find the causality. In this case study, we used a workstation with 2.8 GHz Intel Core i7 CPU, 8.0 GB RAM. We confirmed the verifications are useful for developing software systems.

Related Work There are some other studies for verification of sequence diagrams [KW07] [DD10]. UML 2.0 supports alt, loop, break and opt operators. When developers write sequence diagrams with these operators, CSP or Promela models can be converted from the sequence diagrams using their methods. The main effort of our approach is to provide a method to synthesize state based models from scenario based models even if developer do not use these operators.

Conclusion We developed the SD2CSP tool for verifying sequence diagrams. The tool can verify sequence diagrams where objects can be dynamically created or destructed and the object ids can be passed by message parameters. It converts sequence diagrams to processes in CSP, for checking the their consistency by the model checker PAT. We conducted experiments with a real system in Google using the tool. Our results indicate that verification of sequence diagrams is useful for developing complicated systems.

For future work, we plan to improve the usability of our tool SD2CSP, by extending it with preprocesses of sequence diagrams. For example, we plan an automatic complement of abnormal sequence diagrams. Currently developers need to cover all possible scenarios in sequence diagrams, however, the tool can complement simple scenarios like ”Return error and finish”.

Bibliography

[DD10] L. Dan, L. Danning. An Approach to Formalize UML Sequence Diagrams in CSP. *3rd International Conference on Computer and Electrical Engineering*, 2010.

[KIS13] T. Kaizu, Y. Isobe, M. Suzuki. Refinement and Verification of Sequence Diagrams Using the Process Algebra CSP. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* Vol. E96-A(no. 2):495–504, 2013.

[KW07] A. Knapp, J. Wuttke. Model checking of UML 2.0 interactions. Pp. 42–51, 2007.