COVER: Change-based Goal Verifier and Reasoner

Claudio Menghi[1], Paola Spoletini[2], and Carlo Ghezzi[1]

[1] Politecnico di Milano, Milano, Italy
{claudio.menghi, carlo.ghezzi}@polimi.it
[2] Kennesaw State University, Marietta, USA
pspoleti@kennesaw.edu*

Abstract. COVER is a unified framework that supports the interplay between requirements analysts and software developers. It contracts a bridge between the requirements analyst's and the software developer's artifacts by enabling goal model analysis during software design. The goal model produced by the requirements analyst is kept alive and updated while the system is designed. Whenever the design of the system changes, COVER verifies the new design against the requirements of interest. The verification results are used to trigger a goal model analysis procedure. The results of this analysis can be used by the requirements analyst and the software developer to update the goal model or the design of the system. In this paper, we present the tool support developed for COVER.

Keywords: iterative design, goal model analysis, partial models.

1 Introduction and Motivation

Software development is a complex activity which relies on a sequence of development steps. It usually starts with the requirements analyst collecting the requirements of the system. The identified requirements are used by the software developer as guidelines in the system design, in which a model of the system is produced. The system design is not a straightforward activity. It usually includes a set of development rounds, in which a partial and incomplete model of the system is iteratively refined. In order to satisfy the customer needs, the software developer must ensure that the final, fully specified, version of the system satisfies the requirements of interest.

Researchers' interest has been mainly focused on two separate aspects. On the one hand, researchers have been working on developing techniques and frameworks to help the requirements analyst in collecting and analyzing requirements. KAOS [2], TROPOS [1] and i^* [12] are examples of modeling formalisms used to collect and represent requirements. Over the years many techniques have been proposed to analyze them (e.g., [5]). On the other hand, several algorithms and

* Copyright 2017 for this paper by its authors. Copying permitted for private and academic purposes.

tools to help software developers in the specification of the system behavior and in checking whether the specification satisfies the requirements of the system [8, 7, 10] have been proposed.

One of the main problems of current software development techniques is the lack of integration between these two set of tools, and, consequently, the lack of a bridge between requirements analysts' and the software developers' work. This mismatch particularly affects modern development life cycles which 1. are heavily based on iterative and incremental development processes and 2. require a strong interaction between the requirements analyst and the software developer.

COVER (Change-based gOal VErifier and Reasoner) [11] is a unified framework that enables goal model analysis during software design. Its goal is to integrate models used by the requirements analyst and the software developer to support a continuous interaction between these two actors. This would allow requirements and design to evolve together. COVER enables the continuous verification of the requirements of the system through design iterations and triggers the analysis of the verification results over the goal model. It is a general framework that can applied independently of the chosen goal model and the design formalisms.

This paper presents the tool support (hereafter called either COVERTool or just COVER) used in a specific instantiation of the framework, where the variation of the TROPOS modeling language presented in [9], Modal Transition Systems (MTS) [8], and Fluent Linear Temporal Logic (FLTL) [4] are chosen as a goal model framework, model for the design of the system, and specification language for functional requirements, respectively. The generic approach and a discussion on the limits of the current instantiation can be found in [11].

The rest of the paper is organized as follows. Section 2 gives an overview of COVERTool. Section 3 describes the user experience in using COVERTool. Finally, Section 4 concludes the paper.

2 Overview

COVER[3] is a framework that supports the interplay between requirements analysis and design. It can be used to verify which goals of the goal models are satisfied by the new design every time the developer changes the design of the system, and to verify how the current design satisfies the goals of the goal model, when the requirements analyst modifies the goal model. The overall tool support for COVER is a Java 8 application that uses the Goal Reasoning Tool (Gr-tool) [6] as a design framework of goal models and for the label propagation and the Modal Transition System Analyzer (MTSA) [3] for supporting the developer in the system design. An high-level representation of COVER is presented in Fig. 1. Additional details on the framework can be found in [11].

The framework consists mainly of three main parts: the goal model design and requirements specification, the system design, and the design verification, which enables the goal model analysis.

[3] The tool is available at https://github.com/claudiomenghi/COVER.

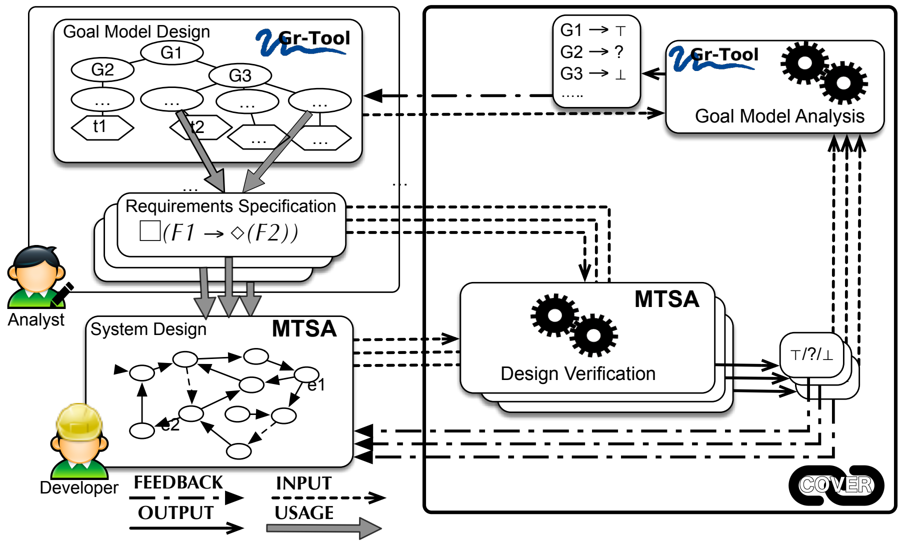


Fig. 1. An high-level overview of the COVER.

Goal model design and requirements specification. The requirements analyst develops a TROPOS goal model for the system using Gr-tool. Goals are refined into requirements, which, in turn, are decomposed into tasks, i.e., the functionalities the system has to provide. The completion of a task is indicated by the developer with an event: the system generates the event if and only if the task of interest has been accomplished. The analyst uses these events to provide a formal description of some of the goals of the system. The analyst chooses the set of requirements to be formally specified in FLTL depending on the system under development. Note that events and FLTL formulae are not part of TROPOS; they are used to link the requirements analyst and developers models.

System design. The software developer produces a high-level model of the system to be in MTSs. Then, she/he iteratively decomposes the system until the behaviors of all of its components are specified.

Design verification and goal model analysis. COVER verifies the new (incomplete/partial) design of the system. It relies on MTSA as a verification tool for MTSs. Since the model is incomplete, the verification procedure may return three different values: $\top$ if the property is satisfied by the current design, no matter how the undefined parts of the system will be later refined; $\bot$ if the property is not satisfied; ? whether its satisfaction depends on the parts which still have to be refined.

The values obtained from the verification of the system design are used to trigger the analysis of the goal model. Each goal of the goal model is associated with an initial value, which specifies whether it is satisfied, possibly satisfied, or not satisfied by the current design. The goals that have not been formalized are considered as not satisfied. A label propagation algorithm is then used to propagate the initial values. The label propagation algorithm in [6] is modified according to the three valued semantic used in the verification of the goal model. The results are used by the requirements analyst and the software developer to change the goal model and the design of the system, respectively. Further details can be found in [11].

3 User experience

We describe how COVER can be used by requirements analysts and software developers.

Requirements analyst perspective. The analyst identifies the requirements of the system in TROPOS, using the Gr-Tool. We assume that the developer specifies each goal by means of an identifier followed by the goal name, i.e., the textual description of each goal matches the following pattern: `IDENTIFIER: GOAL_NAME`. When the final version of the goal model is produced, the requirements analyst formally specifies the requirements of interest. The analyst provides the specification of the requirements in a file with extension `.lts`. This file contains a set of FLTL formulae that formally specify the goals of the system. Each goal starts with an identifier followed by a semi column (e.g., $G1:$) which is preceded by the keyword `assert`. When the formalization of the requirements ends, the requirements analyst forwards the produced file to the software developer.

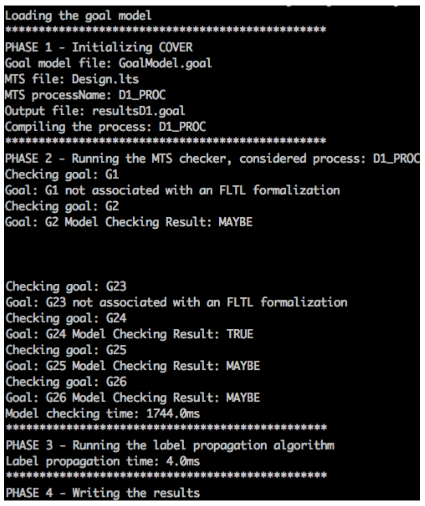
```
java -jar COVER.jar initGoalModel.goal design.lts PROC
    finalGoalModel.goal
```

Listing 1.1. Running COVER

When the software developer produces a new (partial) model of the system, a modified version of the file `.lts`, containing also a behavioral model of the system, is delivered to the requirements analyst. The analyst runs COVERTool by executing the command in Listing 1.1, where:

– `initGoalModel.goal` is the original goal model to be considered;
– `design.lts` is the design to be verified;
– `PROC` is the identifier of the process specified by the MTS contained in the file `design.lts` to be considered by the verification procedure;
– `finalGoalModel.goal` is a file that will contain the goal model updated with the results of the label propagation.

An example of the tool's output is shown in Fig. 2a. COVERTool loads the goal model and iteratively analyzes its goals. When a goal is analyzed, COVERTool extracts the identifier of the goal and checks if the file `design.lts` contains an FLTL property associated with the goal. If a property is specified, it is verified w.r.t. the process `PROC` specified as parameter in the COVER Tool invocation. When all the properties are verified, the label propagation algorithm is executed.

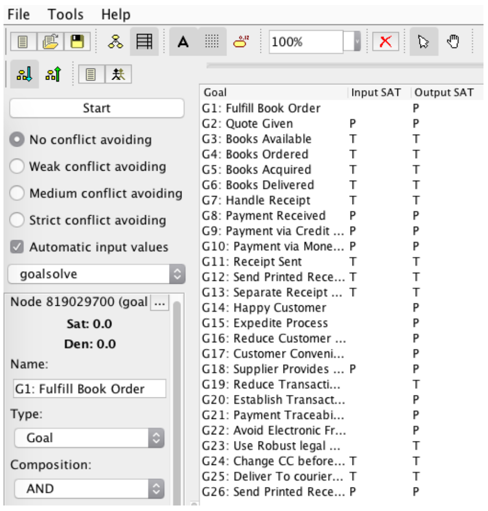


```
Loading the goal model
***********************************************
PHASE 1 - Initializing COVER
Goal model file: GoalModel.goal
MTS file: Design.lts
MTS processName: D1_PROC
Output file: resultsD1.goal
Compiling the process: D1_PROC
***********************************************
PHASE 2 - Running the MTS checker, considered process: D1_PROC
Checking goal: G1
Goal: G1 not associated with an FLTL formalization
Checking goal: G2
Goal: G2 Model Checking Result: MAYBE



Checking goal: G23
Goal: G23 not associated with an FLTL formalization
Checking goal: G24
Goal: G24 Model Checking Result: TRUE
Checking goal: G25
Goal: G25 Model Checking Result: MAYBE
Checking goal: G26
Goal: G26 Model Checking Result: MAYBE
Model checking time: 1744.0ms
***********************************************
PHASE 3 - Running the label propagation algorithm
Label propagation time: 4.0ms
***********************************************
PHASE 4 - Writing the results
```

(a) An example of COVER output.



[✓ No conflict avoiding] Weak conflict avoiding Medium conflict avoiding Strict conflict avoiding

[✓ Automatic input values]

goalsolve

Node 819029700 (goal ...

Sat: 0.0

Den: 0.0

Name: G1: Fulfill Book Order

Type: Goal

Composition: AND

Goal	Input SAT	Output SAT
G1: Fulfill Book Order		P
G2: Quote Given	P	P
G3: Books Available	T	T
G4: Books Ordered	T	T
G5: Books Acquired	T	T
G6: Books Delivered	T	T
G7: Handle Receipt	T	T
G8: Payment Received	P	P
G9: Payment via Credit ...	P	P
G10: Payment via Mone...	P	P
G11: Receipt Sent	T	T
G12: Send Printed Rece...	T	T
G13: Separate Receipt ...	T	T
G14: Happy Customer		P
G15: Expedite Process		P
G16: Reduce Customer ...		P
G17: Customer Conveni...		P
G18: Supplier Provides ...	P	P
G19: Reduce Transacti...		T
G20: Establish Transact...		P
G21: Payment Traceabi...		P
G22: Avoid Electronic Fr...		P
G23: Use Robust legal ...		T
G24: Change CC before...	T	T
G25: Deliver To courier...	T	T
G26: Send Printed Rece...	P	P

(b) Inspecting the output using the Gr-Tool.

Fig. 2. An example of interaction with COVER.

The requirements analyst can graphically inspect the results obtained by COVERTool and contained in the file `finalGoalModel.goal` using the Gr-Tool. For example, Fig. 2b shows an example of results obtained using COVER. The column `Input SAT` contains the results of the verification procedure, where the values T and P specify that the property is satisfied and possibly satisfied, respectively. When a goal is associated with no value, it is either violated or not associated with an FLTL formula. The column `Output SAT` contains the results obtained by running the label propagation algorithm. The requirements analyst can use these results to improve its goal model. Additional details on how these results are used to change the goal model can be found in [11].

The analyst executes the previously described procedure also when she/he already possesses a design of the system and she/he changes its goal model. This allows her/him to verify which goals of the new goal model are satisfied by design.

Software developer perspective. The software developer receives from the requirements analyst the file containing the requirements that the system has to satisfy. The developer produces a behavioral model of the system basing his/her decision on the received requirements and then uses MTSA to verify if the produced model satisfies/possibly satisfies the requirements of interest. MTSA per se does not provide any feedback about the impact of the design decision over the goal satisfaction, but COVERTool in which MTSA is integrated provides this functionality and propagates the results obtained with MTSA back to the goal model. This allows the software developer to analyze the consequences of her/his design choices over the goal satisfaction.

COVERTool is executed using the command specified in Listing 1.1. The obtained results are inspected using the Gr-Tool. The software developer can use the results of the label propagation algorithm to improve its design. Additional details can be found in [11].

4 Conclusion

This paper presented the tool support for COVER, a unified framework that enables goal model analysis during the software development. The goal model produced by the requirements analyst is kept alive during the design of the system. At each refinement round, i.e., whenever the developer produces a new increment or changes something in the model, the new (incomplete) design of the system is verified. The verification results are used to analyze the set of goals of the goal model that are currently satisfied, possibly satisfied and not satisfied.

References

1. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
2. A. Dardenne, A. Van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50, 1993.
3. N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel. MTSA: The Modal Transition System Analyser. In *International Conference on Automated Software Engineering*, pages 475–476. IEEE, 2008.
4. D. Giannakopoulou and J. Magee. Fluent Model Checking for Event-Based Systems. In *Symposium on Foundations of Software Engineering*, pages 257–266, 2003.
5. P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Formal reasoning techniques for goal models. In *Journal on Data Semantics I*. Springer, 2003.
6. P. Giorgini, J. Mylopoulos, and R. Sebastiani. Goal-oriented requirements analysis and reasoning in the tropos methodology. *Engineering Applications of Artificial Intelligence*, 18(2):159–171, Mar. 2005.
7. M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *Programming Languages and Systems*, pages 155–169. Springer, 2001.
8. K. G. Larsen and B. Thomsen. A modal process logic. In *Logic in Computer Science*, pages 203–210. IEEE, 1988.
9. S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos. Integrating preferences into goal models for requirements engineering. In *Requirements Engineering Conference*, pages 135–144. IEEE, 2010.
10. C. Menghi, P. Spoletini, and C. Ghezzi. Dealing with incompleteness in automata-based model checking. In *Formal Methods*, volume 9995, pages 531–550, 2016.
11. C. Menghi, P. Spoletini, and C. Ghezzi. To appear in: Requirements engineering: Foundation for software quality, REFSQ. Lecture Notes in Computer Science. Springer, 2017.
12. E. S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Requirements Engineering Conference*, pages 226–235. IEEE, 1997.