

Goal-Oriented Requirements Analysis for Process Control Systems Design

Islam El-Maddah and Tom Maibaum

Department of Computer Science, King's College London, London WC2R 2LS, UK
{elmaddah, tom}@dcs.kcl.ac.uk

During the last two decades, a lot of effort has been focused on automating the generation of software applications. The automation process can start early, after some manual stage(s) or following (a) previous automatic stage(s). Such automation tools should have the capability of generating executable programs, specifications, or formal requirements, as appropriate. The requirements gathering and checking is considered as the most important phase to eliminate bugs that appear later and may be removed during the design or implementation phases, but with higher cost and effort [10].

Focusing on a narrow family of applications as Process Control Systems increases both the hardware and software reusability in building similar applications of the same family. However, creating Process Control applications needs cooperation between a System Engineer and a Software Engineer. The System Engineer is supposed to have complete information about the operations of the process control system that enables him to provide the requirements to the Software Engineer. Mistranslating the System Engineer's requirements may happen for different reasons and result in incomplete, inconsistent or ambiguous requirements that may result in hazards during the operation of the system.

This gap between the System Engineer perspective level and the formal specification can be filled by means of hierarchical nature of KAOS [2, 3] goal-models that focus on local parts of the requirements and then proceeds to the next parts until completing the whole goal-model. In addition to dividing the effort, the hierarchical nature of the goal-model addresses another important issue, the gradually increasing stages of formality, as the refinement starts from the top of the goal-model until it reaches the maximum at the terminal goal level. This narrows the gap between the System Engineer's perspectives and the level of formal specifications through an adaptation of KAOS.

The GOPCSD (Goal-oriented Process Control systems Design) tool is designed to gather and structure the requirements for Process Control Systems, as shown in figure 1. The tool achieves separation between the Process System Engineer's view and the Software Engineer's. The tool hides the mathematic details of B method [1] from the System Engineer; enabling him to focus only on the operation specifications while the Software Engineer within the B toolkit [7] environment focuses on programming paradigms. Some adaptations have been

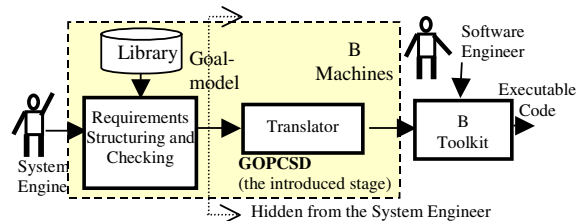


Fig. 1. Inserting an intermediate stage

applied to the method of KAOS in order to yield maximum benefit while minimizing both the required time and the effort to complete the gathering of application requirements for Process Control Systems. After studying different process control case studies, six patterns have been identified: two (alternative and conjunction pattern) of them extending the existing patterns in the KAOS method and four (sequence, disjunction, simultaneous, and inheritance patterns) are new and can be extensively found within the Process Control Systems.

To illustrate the method, we present a case study for a basic gas burner system. The system is constructed, from a gas valve, an air valve, a switch, an igniter, and a flame detector. The flame is kept burning by providing a continuous stream of a mixture of air and gas. In order to prevent any hazards during operation, it is necessary to avoid the gas valve being open while the air valve is closed. The construction of the goal-model of the application starts by importing the application's components and high-level templates from the Library. And then, the application's main goals can be extracted from the application documentation by examining verbs and verb phrases, as indicated in the KAOS method. Each of these identified goals has to be classified as a maintaining, an achieving, a ceasing, or an avoiding goal. Then, an informal description should be defined for each goal, in addition to a formal description, if possible. Such a formal description is a conditional assignment based on the application's variables. The tool provides templates for these four goal types based on Temporal Logic [8]. The next step is to refine these main goals to the existing components' low-level goal-models and to combine them into the high-level goal-model templates. Figure 2 shows one alternative goal-model. The order of combining, refining, importing sub goal-models and formulating the application main goals is not restricted to increase the flexibility of the method. Similarly, the remaining goal-models can be combined together to construct a complete

goal-model that starts with the application's highest-level main goal and ends with terminal goals that are assigned to agents.

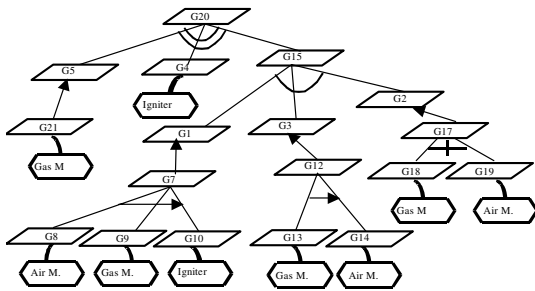


Fig. 2. one alternative solution

The GOPCSD tool allows the user to perform goal conflict [5] and obstacle [4] analyses in addition to checking the completeness and animating the goal-model to validate the application requirements. After checking and validating the goal-model, the tool translates the goal-models into B specifications through the state transition diagram STD shown in figure 3 as an intermediate representation. The goals of the goal-model are equivalent to the transitions within the STD, while the states of the STD are equivalent to pre- and post- conditions of the goals. The non-terminal goals will be represented by indirect transitions, while the direct transitions represent the terminal goals. Some refinement patterns need further treatment, like the conjunction pattern that adds new sub STDs to the main STD. Thus, the terminal goal actions, like changing the output variables values, will be carried out as the application changes its state. It should be noted that, unlike achieve or cease goals, maintain or avoid goals have the final state identical to the initial state. So, states $d3st1$ and $d3st2$ are identical because G4 is a maintain goal. The STD can be translated to B machines in a manner similar to that described in Sekerinski [9]. State variables for each STD will be defined and initialized to the initial states of each STD. Thus, the equivalent B controller machine will have four state variables ($dstate0$, $dstate1$, $dstate2$, and $dstate3$), in addition to the input variables and the output variables). Then, the transitions of each STD will be represented within the operations of the controller machine, as *if* statements. For example, goal G8 will be translated to the following segment of B specification:

```
IF dstate0 = d0st1 THEN
  IF dstate1 = d1st1 & switch = on &
    flame_detected = absent THEN
    dstate1 := d1st3 || open_air_valve ...
```

Here *open_air_valve*, is an operation (to open the air valve within the actuator machine (*air_motor_actuator*)). Thus, at the end, we will have two sensor machines (for *switch* and *flame_detected*), three actuator machines (for *gas_motor*, *air_motor* and *igniter*) and one controller machine. Following this, the B machines can be used within the B Toolkit to generate the implementation version.

Compared to similar approaches [6, 9], the major advantages of the GOPCSD tool are simplicity, reusability, flexibility, and the separation between requirements and specification stages. However, if we consider a modular design for the generated B machines, further research is required to optimize the generated STD, to structure the generated machines or directly create structured ones. On the other hand, we have provided an alternative environment for creating, maintaining, and reusing applications based on the goal-models, thus the need of modular design can be migrated to the requirement level that both the method and the tool of GOPCSD support.

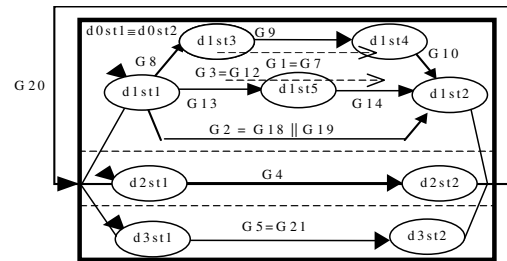


Fig. 3. The generated STD

References

- [1] J. R. Abrial, *The B Book: Assigning Programs to Meaning*, Cambridge University Press, 1995.
- [2] A. Dardenne, A. Van Lamsweerde, and S. Fickas, *Goal-Directed Requirements Acquisition*, Science of computer Programming, Vol. 20, 1993, pp 3-50.
- [3] A. V. Lamsweerde, A. Dardenne, B. Delcourt, and F. Dubisy, *The KAOS Project: Knowledge acquisition in automated specifications of software*, proceeding AAI Spring Symposium series, Track: Design of composite systems, Stanford University, March 1991, pp 59-62.
- [4] A. V. Lamsweerde and E. Letier, *Obstacles in Goal-driven Requirement Engineering*, proceeding ICSE'98 20th international conference on software engineering, Kyoto, ACM-IEEE, April 1998.
- [5] A. V. Lamsweerde, R. Darimont and E. Letier, *Managing Conflicts in Goal-Driven Requirement Engineering*, IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development, Nov. 1998.
- [6] K. Lano, K. Androutsopoulos, and D. Clark, *Structuring and Design of Reactive Systems using RSDS and B*, FASE, ETAPS 2000
- [7] K. Lano and H. Haughton, *Specifications in B, An introduction using the B toolkit*, 1996, Imperial College Press
- [8] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, 1992.
- [9] E. Sekerinski, *Graphical Design of Reactive Systems*, D. Bert (Ed.) 2nd International B conference, Montpellier, France, Springer-Verlag, 1998.
- [10] P. Zave and M. Jackson, *Four Dark Corners of Requirements Engineering*, ACM Transactions Software Eng. and Methodology, 6(1). 1997. 85.