

Tracing Aspects in Goal driven Requirements of Process Control Systems

Islam A. M. El-Maddah and Tom S. E. Maibaum
King's College London
Department of Computer Science
{Elmaddah, Tom}@dcs.kcl.ac.uk

Abstract

Aspects, such as safety, security and productivity play an important role in developing process control systems. These aspects need to be specified at requirements analysis level. Goal driven requirements analysis methods are well suited to trace the different aspects of software applications to the early design level. This paper illustrates how to use the GOPCSD tool in developing aspect-based process control applications. The GOPCSD tool adopts goal driven requirements analysis concepts and has been adapted from the KAOS method to address process control systems. The various aspects of the process control application are modeled using high-level goals within the requirements goal-model. Although these goals are concerned with "soft" issues, the GOPCSD tool guides the user to specialize them to operational/functional sub-goals, assigned to (combinations of) appropriate agents. This can prepare the stage for the consistency and completeness checks offered in the GOPCSD tool to bring the different aspects together. The user will be able to reason about the why of the requirements units, as well as validate the overall operation of the process control application. Finally, the tool will automatically translate the corrected requirements into a B specification, which traces the early requirements aspects to the initial high-level design.

1. Introduction

The software industry has insinuated itself into many applications, reaching a considerable level of maturity in many areas. Consequently, the struggle in the software sector is turning towards the quality of the produced applications. Apart from product operation, issues such as how much safety, usability, security, mobility, and reusability the software products offer are gaining more attention from the client's perspective. These quality aspects make one application preferable to another and, hence, encourage the client to choose it. As noted in [11], more effort should be paid to the quality aspects of requirements for the application. Hence, the software provider needs to pay equal attention to the quality of achieving client's functions and the functions, themselves. This indicates the value of integrating quality attributes and operation of the

software applications as early as at the requirements and user needs levels.

Delaying the aspect requirements later to the design and implementation levels, without explicitly demanding them at the requirements analysis level, may result in potential conflicts with the original requirements, as well as operational elements of the design. In particular, this later stage of the development lifecycle may present difficulties in changing/altering the original requirements, which may in turn result in making the client unhappy. In addition, agreeing with the client only on the functional requirements may be likened to contracting one application and developing a different one. In fact, the functional requirements should be regarded as what the system should perform, whereas the quality aspects should affect the manner of this performance. For example, in a lift system, safety aspects can restrict the movement of the cabinet if the door is not closed. Thus, the specification of the functions and the restrictions enforced by qualitative aspects should be better specified at the same level. So, not only should the aspects not die, as in [23], at the implementation stages, destroying traceability, but aspects should also be alive from the beginning, at the user needs and system requirements stage.

In the field of process control systems, quality aspects include safety, usability, security and economic. Neglecting one of these aspects in control applications would result in unsafe, insecure, or very expensive solutions, which will usually be rejected by the client. Although two control applications can perform the same functions, they may still differ in how much safety they offer, how easy it is to use them, or how much power they consume. This can make one of them more successful than the other.

There has been considerable interest in dealing with aspects at the level of programming languages by supplying explicit language constructs to implement aspects: AOP [14], AspectJ [30], PROSE [27], [3], [5], ARCaDe [28], [29]. However, at the requirements and specification level, aspects are represented using appropriate language constructs, or are implicit in specifications of the safety, livens and fairness

properties, as in B [1, 4], VDM [13], SMV [24] RSDS [18] and SCR [12].

In [7], aspects within requirements goals were monitored at runtime; the implementation code had two main parts: one to fulfill the operational goals of the system and the other to monitor the aspects and their requirements. In Planguage (a programming planning language), Gilb in [11] has suggested some quantitative scales to be used to measure the quality aspects.

To represent the aspects in the requirements level, the requirements model has to be able to trace these aspects from the level of client needs down to the early specification level, and then to the implementation level [6]. Moreover, the developed method should provide reasoning about the why of the various parts of the requirements, to validate the aspect-based requirements model as in KAOS [15]. In addition, an aspect-based development should provide an opportunity to evolve the requirements or change them later in response to the results of changing client needs and/or the results of analysis.

One of the requirements approaches that devotes a high level of attention to traceability and understandability is goal driven methods, such as KAOS [15] and TROPOS [25]. In these methods, the requirements are represented in terms of hierarchies of goals called goal-models. Each goal represents an identified requirement, possibly an aspect, like safety, security or function. The main goal of each goal-model specifies the overall application and usually is refined to sub goals that represent different parts, modes, and aspects of the applications. The refinement process ends at terminal level, where each terminal goal is simple enough to be assigned to an agent to carry it out *as an operation*. These operations may be assigned to agents such as software components, humans or hardware devices. Since all higher-level goals are eventually realized as a combination of terminal goals operationalised by (combinations of) agents, higher-level goals representing aspects are transformed to operational goals realized by agents. This transformation of aspects to combinations of operational goals realized by combinations of agents is the key to aspect-based design provided by (some) goal oriented requirements methods like KAOS and GOPCSD.

As a result, the aspects can be traced down from a high-level goal to the terminal goal level, where they can be translated to early design form, as a B specification [1], which is acknowledged as a suitable formal tool for designing reactive systems. This traceability from requirements to early design offered by the goal-models ensures Aspect Oriented Software

Development (AOSD) will fulfill its goal of being a software engineering method, as noted in [5].

Although the B method suits the design of process control systems, the formal logic and sophisticated mathematics of its formal language have hindered both its usability and understandability by users, such as process systems engineers, and restricted its use to a small part of the software engineering community. This demonstrates some interference of concerns between the systems engineer, as the client, and the software engineer, as the service provider. More informality should be allowed to the client to be able to express these quality aspects and the requirements method should not depend on deep knowledge of some specific design and implementation method and its possibly alien and unfamiliar formalisms and notations. Hence, some automated mapping should translate the agreed requirements to a formal design model, as noted in [21]. Thus, we were motivated to develop the GOPCSD (Goal Oriented Process Control Systems Design) tool [10] that adapts the goal driven requirements analysis method of KAOS [15, 16], but after some significant adaptations to fit the nature of process control systems.

This paper consists of six sections. In section one, we introduced the research area. In section two, we briefly describe the GOPCSD tool, especially in relation to developing aspect-based requirements. In section three, we illustrate how to create aspect-based goal-models in GOPCSD, using some process control examples. Then, in section four, we show how to bring the different aspects together using the checks and tests offered in the GOPCSD tool. In section five, we use the animation utility offered by the tool to validate the corrected requirements after bringing the aspects together. Finally, in section six, we draw conclusions and suggest future directions for further work.

2. The GOPCSD Tool

The GOPCSD tool [10] has been developed to manage rationally the gap between the process control systems engineer's perspective, as the client, and the formal specification level, supplied normally by a software engineer. To accomplish this; we have developed an integrated requirements development environment, where the process control requirements can be constructed using a provided library, structured in terms of hierarchies of goals, and then checked, corrected, validated and finally automatically translated to a B formal specification. In the following sub sections, we briefly describe the requirements elements to represent process control applications in

the GOPCSD tool and the development phases needed to complete the generation of a formal specification in B corresponding to the corrected requirements.

2.1 Requirements Elements

The goal-oriented requirements of any process control system will be represented within the GOPCSD tool by the following elements:

2.1.1. The Components. In process control systems, components represent the physical parts of the applications, such as valves, robots, and deposit belts. The detailed specifications of each component, including its variables, agents and goal-models, are stored in the GOPCSD library. The systems engineer can create/edit the component details using the GOPCSD library manager.

2.1.2. The Variables. Variables are considered as an essential part of formalizing the user requirements. In the GOPCSD tool, the application's global state is usually described by a set of variables. Each of these variables has one of three types: input, output or intermediate. In the GOPCSD tool, the variables are associated with the high-level goal-model templates or the components, each of which the user can import from the library; however, the tool user can also create, edit, and delete variables from the application design space.

2.1.3. The Agents. Agents are the objects that control the application parts and its local environment. Some of the agents can be part of the application to be built, like software interface programs for hardware parts, or, alternatively, they can be existing programs or hardware devices that will be responsible for accomplishing defined goals to fulfil the overall application operation. Agents can have one of the following three types: device, software and human. The main source of agents is via the user's importing of components from the library. But, if it is required to declare agents apart from those associated with the components, the user can create, edit, and delete them from within the GOPCSD tool environment.

2.1.4. The Goal-models. Goal-models constitute the main segments of the structured requirements; they represent the user requirements as a hierarchy of goals. Each goal-model starts with a main goal that has general scope; this main goal is usually refined to a number of sub-goals describing sub-parts, different

aspects, or operation-modes of the application. Each of the goals within the goal model represents a qualitative or operational requirement; the tool supports informal descriptions of goals as well as formal descriptions based on temporal logic [22].

2.2 Development Phases

The GOPCSD tool covers the early development stages. It refines and formalises the abstract user's needs to functional and formal specifications. After the user corrects and validates the requirements, the tool automatically generates a B specification. The tool has three development phases as follows:

2.2.1. Phase One. In this phase, the GOPCSD tool guides the user to construct the requirements and structure them in terms of goal-model hierarchies. The tool offers the user two options to choose from or to integrate in constructing the requirements: to import from the provided library (this library is dynamic in the sense that the user can create components and templates and store them in the library for future use), or to create the requirements entities from scratch using the tool's utilities. After importing the components and templates or creating them, the next stage is to refine the incomplete goals and to combine the separate high level goal-models, representing abstract operational and qualitative goals, with operational goal models corresponding to components, to arrive at a complete model where each terminal goal is assigned to an agent as an operational goal. When the user manages to construct a single goal-model specifying the entire application, the second phase of correct and debugging the goal-model can start.

2.2.2. Phase Two. This phase checks the completeness and consistency of the requirements. The tool provides various checks to remove the major part of the requirements bugs. In addition, an animation utility is offered to validate the requirements by executing the goal-model corresponding to the application. Thus, the application performance can be portrayed for the client before generating formal specifications and implementations. The tool enables the user to stimulate the system by changing the values of input variables and mimicking faults and delays in the different components.

2.2.3. Phase Three. This is the third and final phase of generating formal specifications. After the goal-model has been checked and modified, the tool can

automatically generate general “use case”-like operations that each has an associated pre- and post-condition and a responsible actor (one of the application’s agents) to perform the operation. The GOPCSD tool also generates a B formal specification. The generated B machines will be documented using the informal description of the requirements elements supplied by the systems engineer.

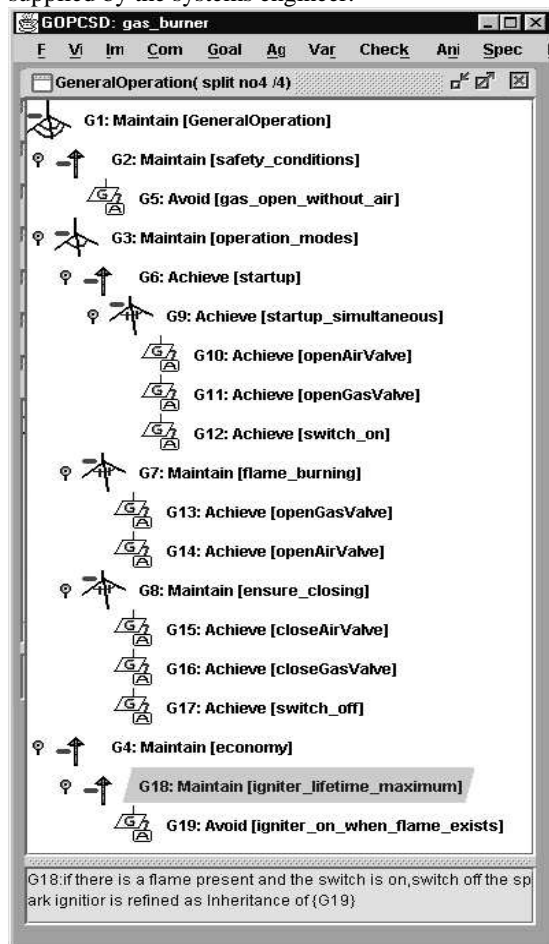


Fig. 1, the goal model of the gas burner system

2.3 The GOPCSD Tool Support for AOSD

The various aspects of the process control application requirements will be represented as high-level conjunctive goals. Aspect keywords, such as safety, productivity, etc., will be written explicitly within the informal description of the high-level goals. The sub goals of one aspectual goal will inherit the aspect from their parent goal. The non-aspectual high-level goals will be concerned with the normal operation of the system, and will usually be grouped under one high-

level goal (see goals G3 in fig. 1 and G5 in fig. 3). Each high-level aspectual goal will be refined to a level in which the goal representing the aspect can be formulated as a combination of operations that demands an operational scenario to be undertaken by a combination of agents.

After completing the construction of the goal-model, the different aspects of the requirements will be brought together through the consistency and completeness checks. The tool provides utilities for reasoning about why and how for the different goals, including aspect related ones, of the goal-model. Thus, an early correction path is achieved as well as an informal means for better understanding of the constructed requirements model.

To validate the constructed requirements model, the GOPCSD tool provides an animation utility, which can reason about the why of taking the current actions in terms of aspect goals, cycle by cycle, during the symbolic execution. Finally, the tool automatically generates a B specification corresponding to the corrected requirements, which is documented (using the informal description segments of the goals) to be able to trace the aspects down to the implementation level.

3. Creating the Aspect-based Requirements

To achieve the single requirements model, which is the aim of the GOPCSD method, from the multiple aspect requirements, the construction of the requirements specifications may use one of two variations. In the first one, the specification has to consider the different aspects all at once. This (eventually) entails that the defined operations involving pre- and post-conditions must combine to affect the different aspects. For example, in a production cell, the robot arms should be moved in a safe way, while also ensuring that the service time for each metal blank will be as small as possible.

In another possible variation, a separate requirements view concerning each aspect can be constructed; and, then, the requirements from the different views can be examined to ensure they do not prescribe any incomplete nor inconsistent behavior, ensuring the views are integrated. For example, regarding the robot arm movement, there will be different requirements from the point of view of safety, productivity and operation. Then, using conflict and completeness analyses, some situations will be highlighted, where goals relating to different aspects will compete. Thus, by modifying the various aspect related goals, after the problems pointed out by

consistency and completeness checks, the requirements model will automatically adapt itself to weave the various aspects together.

The second approach has the promise of less effort being expended for the design of each aspect view (due to simpler models of each aspect), as well as similar effort being required to test the completeness and consistency of the requirements overall. Hence, as noted in [2] to decrease the effort required to put the aspects together, we adopt this second approach to group the requirements by aspects at the high-levels within the goal-model. The requirements model will be then elicited by refining the initial user needs into detailed system requirements.

In one case study, we examined a gas burner system, as studied before in [17]. The burner system has safety requirements (to control the gas concentration in the area around the burner to avoid explosion) and economy requirements (to increase the lifetime of the igniter). The GOPCSD tool guides the user to elicit the requirements in a goal-model, which has three main goals G2, G3 and G4 concerning safety, operation and economy, respectively, as shown in fig. 1. Each of these goals has been refined to sub-goals concerning local parts or stages of the burner operation. Since the economy is considered as a soft goal (difficult to be formalized or measured), we specialize the economy goal (G4) to a sub-goal (G18), which maximizes the igniter lifetime. Similarly, because it is difficult to formalize and represent all factors affecting the lifetime of the igniter, we specialize goal G18 to G19, where we formulate a goal of avoiding switching the igniter on if the flame already exists. What can be done to effectively manage the lifetime of a physical component is exactly the kind of knowledge that we would expect a process systems engineer specialized in burner systems to possess.

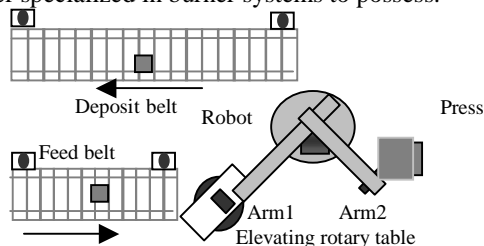


Fig. 2, the production cell

For safety requirements, which are again considered as a soft goal (G2), we specialize to an instance of the safety goal that concerns itself with the gas concentration around the burner (G5). In goal G5, it was then easy to formalize the requirement about the concentration of the gas being controlled by avoiding

opening of the gas valve while the air valve is closed. Again, this is likely to be the kind of specialist knowledge possessed by the process systems engineer. Thus, we were able to formalize some issues related to soft concepts/“non-functional requirements” by using the specialization sub-goaling mechanism of GOPCSD. All other sub-goaling mechanisms in GOPCSD require a logical equivalence between the goal and its set of sub-goals.

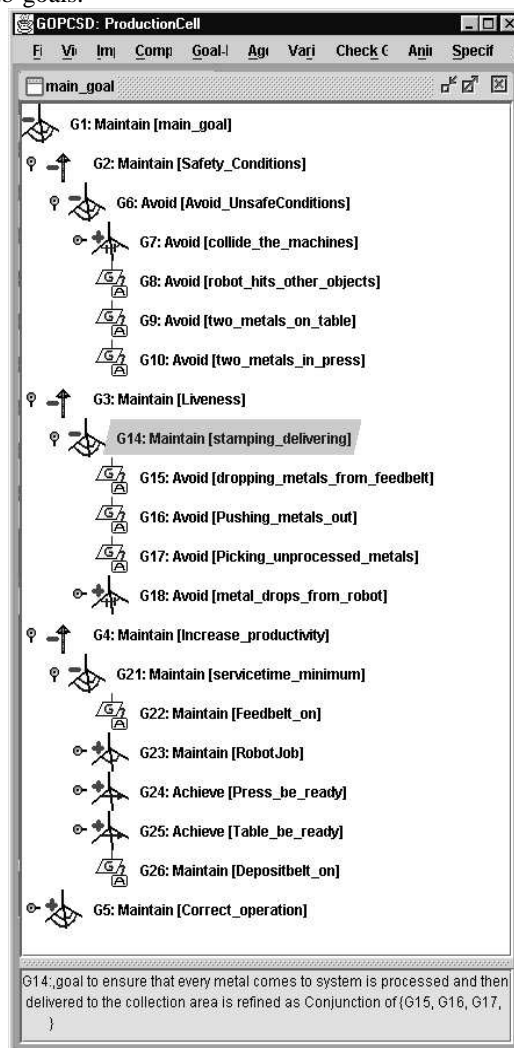


Fig. 3, the refined goal-model of the production cell

In another case study, we examined a production cell [20], as shown in fig. 2. The production cell has a safety goal (to avoid collision between the different machine parts), a productivity goal (to increase the throughput of the system) and a liveness goal (to insure every blank metal coming through the feed belt will be stamped and delivered to the deposit belt). We model each of these aspects by one high-level goal, as shown

in fig.3. Goals G2, G3, G4 and G5 represent the safety, liveness, productivity (throughput) and operational goals, respectively. Each of these goals is refined, separately, as shown in fig. 3.

Although maintaining safety conditions for the production cell is considered as a soft goal that has many factors, some of which are difficult to determine and/or control, we specialize the sub goal G6 (concerned with the avoidance of dangerous and controllable situations) from the safety goal G2. Goal G6 is designed to avoid some predicted circumstances that may arise during run-time and are detectable and avoidable by the control system. Again, the knowledge of the process systems engineer is crucial in identifying the right specialization to apply. Hence, goal G6 is refined as a number of conjunctive goals that are designed to avoid the machines colliding with each other (G7), avoid hitting any objects by the robot arms (G8), avoid having two metals on the table tray at the same time (G9) or inside the press at the same time (G10). Goal G7, avoiding collision between the different machines can be refined as -goal G11-avoiding the table hitting robot arm1, and -goal G12-avoiding the press hitting robot arm1 and -goal G13-arm2. Each of these sub-goals is tracing the safety aspect and can be formally specified and assigned to an agent, such as a robot arm or a table motor, as a set of operational goals.

Like safety, the productivity issue has a soft comprehensive goal G4. The productivity of the cell is affected by various factors. Some of these factors are unmanageable, whereas others are uncontrollable by the designed application. Thus, we create a more rigid/harder goal (G21) that resembles the accessible factors the control application can operationally perform to decrease the service-time for each individual blank metal. So, we specialize G4 into an instance G21 concerning with the reduction of the service time as a functional factor to increase the productivity. Goal G21 can be refined in terms of sub goals reducing the service time in the local parts of the production cell. A possible refinement could be as follows: G22, maintain a continuous stream of incoming blank metals; G23, keep the robot ready to serve either the press or the table; G24, keep the table ready to receive new blank metals; G25, keep the press ready to process a new metal; and G26, keep delivering the processed metals. Although these sub-goals are tracing a non-functional aspect, each of them is considered on its own as a high-level functional goal that can be refined systemically into operational sub-goals.

Similarly, the liveness goal G3 can be refined into operational goals, as can the operational goal G5.

Note that again we have used a combination of specialization of operational/non functional goals to functional ones, and the usual refinement of operational goals, to achieve the completion of the overall goal model for the application

Designing each aspect view separately, as in [2], can be considered an advantage because it is much easier to concentrate on the different views separately rather than a single combined view. Furthermore, allowing different views for each aspect encourages a group of systems engineers to develop the requirements model in parallel. In addition, the aspect goals and sub goal-models can then be reused from similar applications and/or previous versions of the same application.

4. Putting the Various Aspects together

Bringing the different aspects together could be achieved dynamically at run-time or in advance. As in [27], to dynamically weave the different aspects, the aspects should be traced into the executable segments of the program. Thereafter at runtime, the segment that traces the most important aspect to fulfill amongst the activated segments should be executed. We could equally apply this approach to goal driven requirements by ranking the goals, according to the aspects, which they are tracing.

An alternative direction to put the aspects together instead at runtime entails some analysis to be carried out to put the requirements composed from the different aspects together before the final implementation stage. This analysis should examine the various situations that can occur during run-time.

Although it is difficult to examine all possibilities that may occur during run-time, especially at requirements stages, the second approach can still be considered better than the first, from the validation point of view, since the decisions about what to do when requirements of different aspects conflict were validated at the requirements stage and not left for resolution when the run-time system was being created.

In particular, this latter approach should be the right one if the aspects do not have absolute rankings over each other for all run-time configurations, as noted in [29]. For example, in some circumstances productivity decision should come before operational ones, while in other circumstances the opposite situation may occur.

Thus, we adopt the idea of performing some analysis at the requirements stage to predict the

situations when the different aspects can undermine each other. The GOPCSD tool allows the user to perform consistency (goal-conflict analysis) and completeness checks for this purpose.

Conflict happens within goal driven requirements when different goals are simultaneously fired and attempt to control the same parts of the components of the system. To be able to remove such cases, the requirements need to undergo consistency checks to ensure such conflicts will not happen during run-time. The GOPCSD tool provides consistency checking [8] based on testing the various runtime configurations, to detect whether the active goals under each of these configurations prescribe inconsistent behaviors, as in [16].

For example, having performed consistency check on the complete goal-model of the production cell (shown in fig. 3), it shows that goals G40 and G11 are conflicting. Goal G11 restricts the rotation of the robot if its first arm is extended until the rotary table moves down to avoid hitting it, whereas goal G40 (grandchild goal of goal G23, to increase the productivity of the cell by moving the robot if it is free to wait for the next expected job) prescribes the rotation of the Robot towards the press to be ready to pick up the metal, which is being pressed at that moment. Knowing that goal G40 has a productivity aspect (increasing the throughput) and goal G11 has a safety aspect, we should modify the pre-condition of the productivity goal G40 to make sure the two goals will not be active at the same time. The pre-condition of goal G40 can be modified to be mutually exclusive with respect to the pre-condition of goal G11. Thus, goal G40 will be restricted to be active only when the robot arms are retracted. Thus, one of the safety aspect goals (G11) affected the way the productivity goals work. Similarly, other conflicting goals could be changed, paying attention to the importance of each aspect for each individual conflict.

The incompleteness and indeterminism of the requirements may lead to situations where the system prescribes strange or unplanned run-time behaviors. For example, the control system may toggle between two states (or possibly more) attempting to address different aspects. Although the control system actions are consistent, they might exhibit strange behaviors because of the indeterminism and incompleteness concealed within the unchecked requirements. Hence, the GOPCSD tool offers a completeness check based on the completeness of software requirements specification (SRS) [6], which discovers the cases where the requirements prescribe indeterminism or

absence of control. Consequently, the tool provides suggestions for the user to resolve such cases.

Another example to show how completeness analysis can be employed to merge the various aspects is provided, as follows. According to normal requirements specifications of the gas burner system, if the gas valve is open, it closes whenever the air valve is closed to maintain safety conditions, goal G5. On the other hand, when the user switches the burner on, the gas valve opens (goal G11: start the system up when requested) and the control system should ensure the two valves are open to maintain the flame burning (G13 keeps the gas valve open to maintain the flame burning). These three goals can prescribe a behavior of indeterminacy, when air valve is closed and the switch is open. (*switch_switch_state = ON and air_valve_state = CLOSED*). This results in a state, which toggles between opening and closing the gas valve. It is essential to remove such a situation and similar situations. The solution is to modify the pre-conditions of these goals to make sure they are disjoint, and then the behavior will be deterministic. This may be achieved by enabling the safety goal G5 to close the valve only when the switch is off. This is probably unacceptable because the unsafe situation can occur during starting up, when the switch is on. The other choice, which is more logical, is that the gas valve will not be open unless the air valve is already open, i.e. to restrict goals G11 and G13 by adding (*air_valve_state=OPEN*) in their pre-conditions. Having changed the pre-conditions of goals G11 and G13, the safety and operational goals are merged together, without conflicts. (Of course, having changed the goal-model, conflict and completeness checks should again be applied.)

The systems engineer will almost certainly be in a better position to take responsibility for making these decisions before the GOPCSD tool generates B formal specifications and leaving the stage prepared for a software engineer to get involved in developing the control application, within the B toolkit [4] or other similar environments.

5. Validating the Requirements

An important step is to validate the requirements, especially after modifying them to combine the different aspects. The GOPCSD tool provides an animation utility, which symbolically executes the goal-model. The traces of execution cycle by cycle contain the applied events (changes in the input variables) and the list of the activated goals within the goal-model, which trace the different aspects from the high-level

abstract goals to the terminal functional goals. Thus, the user can be acquainted with the current state of the system and reason about the why for system actions. Because each goal is tracing one aspect, the user can be familiar with how the various aspects will work together to fulfill user needs. This increases the understandability of the requirements model. The generated formal specification usually does not achieve such a high level of understandability (a main drawback of formal methods), at least by the normal systems engineer.

In addition, the animation utility allows the user to change the system state and observe the activated goals. This can be effectively used to check whether and how the process control system can recover from unsafe/insecure states. Serving as an early conceptual prototype, the animation utility can guide the systems engineer to fine tune and enhance the requirements.

6. Conclusions

The requirements stage of software development is acknowledged to be the main factor that defines the relative success of software application projects [26, 31]. Thus, the quality aspects should certainly be represented in the requirements stage. This motivated us to design the GOPCSD tool to trace the client's needs related to aspects down to the formal specification level.

The GOPCSD tool adopts the main concept of goal driven requirements analysis of KAOS. Within GOPCSD, the user can create aspect requirements, separately as sub goal-models, and then to combine them using completeness and consistency checks. The corrected requirements will be automatically translated into a B specification by the tool.

The GOPCSD tool does not restrict the user to begin the development lifecycle with a perfect requirements design. It guides its user to reach this stage after a feedback loop of checking and enhancing the requirements. This issue can be clearer in medium- and large- scale applications, where constructing a complete and consistent requirements model at the beginning is usually difficult to achieve.

The GOPCSD tool enables the user to express the productivity (throughput), liveness, safety, economy and operational aspects as conjunctive high-level goals. The requirements completeness and consistency checks can be employed to combine the different aspects in the same manner they are used to remove the requirements bugs. This will not entail more effort on the user side. The user may possibly have an easier and simpler view of each aspect rather than a single view that combines

the aspects. The tool guides the user to develop the application faster in this respect, as recommended in [2]; it provides the integration of the different views using the conflict, completeness and, possibly, the animation utilities. This reduces the effort required by the systems engineer as well as not requiring a high-level of expertise in the process control field.

The way the different aspects are separated makes it easier for the user to decide which goal to weaken or strengthen when resolving conflicts or incompleteness; in addition, it helps to increase the modifiability, traceability, and augmentability [6] of the entire requirements model of the process control application.

Further research should be directed towards evaluating the various aspects while symbolically executing the requirements, as noted by defining quality metrics in [11] and incorporating them via the animation utility. This can guide the client/implementer to have an early measure for how much each aspect is achieved in particular requirements specification solutions.

Another direction for future work is to see how we can remove the discovered conflict between the aspect related goals by prioritizing them, as noted in [16]. When more than one goal is fired simultaneously, the one with highest priority will be allowed to perform its action. This direction reduces the effort required by the user to modify the pre-conditions. However, further effort may be directed towards alerting the user to what will happen during run-time and also enabling him/her to overwrite the default priority system in some configurations. Thus, this can maintain the aspect sub goal-model ready to be reused in other versions of the application.

References

- 1 J. R. Abrial, "The B Book: Assigning Programs to Meaning", Cambridge University Press, 1995.
- 2 M. Aksit, Separation and Composition of Concerns in the Object-Oriented Model, ACM Computing Surveys, volume 28, p 148, 1996 Position paper for the ECOOP '96 adaptability in OO software development workshop, 1996
- 3 M. Aksit and B. Tekinerdogan, Aspects-oriented Programming using Composition-Filters, in Object-Oriented Technology, S. Demeyer and J. Bosch (Eds.), ECOOP' 98 Workshop Reader, Springer Verlag, pp. 435, 1999.
- 4 B Core UK limited (1998), B Toolkit <http://www.b-core.com/btoolkit.html>
- 5 S. Clarke and R. Walker, Towards a Standard Design Language for AOSD, Proceedings of the 1st

- international conference on Aspect-oriented software development, 2001
- 6 Alan M. Davis, *Software Requirements: Objects, Functions and States*, Prentice Hall PTR, 1993
 - 7 A. Dingwall-Smith and A. Finkelstein, "Monitoring Goals with Aspects," University College London, Dept. of Computer Science August 2003.
 - 8 S. Easterbrook, "Resolving Requirement Conflicts with Computer-Supported Negotiation", In *requirement engineering: social and technical issues*, M. Jirotko and J. Goguen (Eds.) Academic Press, 1994, pp 41-65.
 - 9 I. A. El-Maddah and T. S. E. Maibaum, *Goal-oriented Requirements Analysis of Process Control Systems*, Proc. First ACM & IEEE International conference on Formal Methods and Models for codesign (MemoCode), France, 2003
 - 10 I. A. El-Maddah and T. S. E. Maibaum, *Goal-oriented Process Control System Design Tool*, Tool Exhibition of FM03, Italy, 2003
 - 11 T. Gilb, *Competitive Engineering, a Handbook for Systems and Software Engineering Management using Planguage*, 2003, Addison-Wesley
 - 12 C. Heitmeyer, J. Kirby, Jr., B. Labaw, and R. Bharadwaj, SCR*: A toolset for specifying and analysing software requirements. In Proc. Computer-Aided Verification, 10th Annual Conf. (CAV' 98), Vancouver, Canada, 1998
 - 13 C. B. Jones, "Systematic Software Development using VDM", Englewood Cliffs, NJ: Prentice-Hall, 1990.
 - 14 G. Kiczales, J. Lamping, A. Menhdhekar, Chris Maeda, C. Lopes, Jean-Marc Loingtier and J. Irwin, *Aspect-Oriented Programming*, Proceedings European Conference on Object-Oriented Programming, volume 1241, Springer-Verlag, pp 220-42, 1997
 - 15 A. van Lamsweerde, A. Dardenne, B. Delcourt, and F. Dubisy, "The KAOS Project: Knowledge acquisition in automated specifications of software", proceeding AAAI Spring Symposium series, Track: "Design of composite systems", Stanford University, March 1991, pp 59-62.
 - 16 A. van 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.
 - 17 K. Lano and A. Sanchez, *Design of Real-time Control Systems for event driven Operations*, Formal Method Europe, LNCS vol. 1313, Springer-Verlage, Berlin, Germany, 1997
 - 18 K. Lano, K. Androustopoulos, and D. Clark, *Structuring and Design of Reactive Systems using RSDS and B*, FASE, ETAPS 2000
 - 19 N. G. Leveson, M. P.E. Heimdahl, H. Hildreth, and J. Reese. *Requirements Specification for Process Control Systems*. IEEE Transactions on Software Engineering, Vol. SE-20, No. 9, pp. 684-707 (September 1994).
 - 20 C. Lewerentz and T. Lindner, *Case Study "Production Cell" a comparative study in formal software development*, FZI Karlsruhe, 1995
 - 21 T. S. Maibaum (2000) "Mathematical Foundations of Software Engineering: a Roadmap", future of Software engineering, Limerick, Ireland
 - 22 Z. Manna and A. Pnueli, "The Temporal Logic of Reactive and Concurrent systems", Springer-Verlag, 1992.
 - 23 F. Maththijs, W. Joosen, B. Vanhaute, B. Robben and P. Verbaeten, *Aspects should not die*, position paper at the ECOOP'97 workshop on Aspect-Oriented Programming, 1997
 - 24 K. L. McMillan *Symbolic Model Checking: An approach to State Explosion Problem*, Kluwer Academic, 1993
 - 25 J. Mylopoulos and J. Castro, *Tropos: A Framework for Requirements-Driven Software Development* In J. Brinkkemper and A. Solvberg (eds.), *Information Systems Engineering: State of the Art and Research Themes*, Lecture Notes in Computer Science, Springer-Verlag, p. 261-273, June 2000
 - 26 B. Nuseibeh and S. Easterbrook, *Requirements Engineering: a Roadmap*, Future of Software Engineering, Limerick, Ireland, 2000
 - 27 A. Popovici and T. Gross and G. Alonso, "Dynamic weaving for aspect oriented programming", Proceedings of the 1st International Conference on Aspect-Oriented Software Development, 2002
 - 28 A. Rashid, A. Moreira, and J. Araujo, *Modularisation and Composition of Aspectual Requirements*. 2nd International Conference on Aspect-Oriented Software Development. ACM, 2003, pp 11-20
 - 29 A. Rashid, P. Sawyer, A. Moreira and J. Araujo, *Early Aspects: A Model for Aspect-Oriented Requirements Engineering*. IEEE Joint International Conference on Requirements Engineering. IEEE Computer Society Press, 2002, pp 199-202
 - 30 Xerox Corporation. *The AspectJ Programming Guide*. Online documentation, 2001, <http://www.aspectj.org/>
 - 31 P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering", ACM Trans. Software Eng. And Methodology, 6(1). 1997. p 85