Goal Driven Requirements Analysis of Process Control Systems

Islam A. M. El-Maddah

King's College London, Department of Computer Science, London WC2R 2LS, London, UK, Fax +4402078482851
elmaddah@dcs.kcl.ac.uk

Abstract. This paper illustrates the component-based development of process control requirements using the GOPCSD (goal-oriented process control systems design) tool. In GOPCSD, we use components, variables, agents and goal-models to model and structure the requirements of process control systems in terms of goal-models; the supporting tool offers a library of frequently used components and general templates describing components, variables, agents and goal-models to reduce the effort required to build applications. A production cell case study is examined to illustrate the power of the GOPCSD tool in interactively guiding the systems engineer to build, analyse and correct the goal-oriented requirements and to catch requirements bugs as early as possible. After correcting the requirements, the tool automatically generates a B specification for the corrected requirements. Finally, we compare our method with related methods and draw conclusions and directions for future work.

1 Introduction

Automating the operation of process control systems has been attracting increased attention and development effort. Various methods and tools have emerged to cope with the sophisticated nature and the component-based features [11] of the process control systems. This demands developing systems based on integrating sub-systems and extending existing systems rather than building systems from scratch.

Process control systems such as production cells, chemical plants and automotive systems share common safety and quality aspects apart from the operational aspects. Such aspects constrain the behaviour of each component as well as the interaction between the different components within the system. Therefore, a need for a verifiable and provable implementation, which conforms to a precise, correct, safe, secure, complete, and consistent design, has emerged.

Formal methods have achieved considerable success in developing safety-critical reactive systems [19] in general, including process control systems [7, 8, 9, 10, 13, 16, 18]; this success comes from the fact that the formal software specifications of the systems to be developed can be traced to the implementation level and/or the implementation can be partially or completely generated from the specifications, using formal methods tools. In particular, the B formal method [1] fits the nature of process control systems since it has abstract machine notations (AMN) that can model systems as abstract state machines built from components, each of which consists of states defined as a set of variables and state-transitions defined by a set of operations that change the component state through assigning different values to the variables constituting the state. Each component can be specified in a separate module called a machine. The B language allows interaction between the different machines to build hierarchical systems [9, 10, 18]. This language is supported by software tools such as the B toolkit [2], which offers various utilities, like proof obligation generation, animation, and generating the code for an implementation.

However, the requirements provided to the formal method as an input usually exhibit incorrect, incomplete and/or inconsistent behaviours. More attention should be paid to this initial stage, where the informal and abstract user needs have to be correctly translated to formal and precise high-level designs. Moreover, this translation is usually a potentially shared and interference prone responsibility of the systems engineer as the client, who usually has limited knowledge about programming languages or formal methods, and the software engineer as a provider. This interference of concerns arises when the informal and abstract requirements need to be refined and formally modelled as formal specifications, as well as when the specification prescribes erroneous or inconsistent behaviour and needs to be modified. And hence, the corresponding requirements need to be modified.

Thus, because of the importance of the requirements stage [17], as the main factor in achieving successful software applications [23], we were motivated to focus on the requirements gathering, enhancing and correcting stage and to decrease the gap between the systems engineer's perspective and formal specification level, by providing a tool that is easy to use by the systems engineer and automatically generating formal specification to be further processed by the software engineer.

We aimed at choosing a model that can be easily understandable by the systems engineer and provide effective means for tracing the user needs to the requirements as well as transforming the requirements to the specification level in a traceable way. Hence, we adopted the KAOS (knowledge acquisition of automated specification) method [20]; however, the KAOS method is still too general for the field of process control systems; the method needs to be specialized, extended, and refined in light of the nature of its users, usually measurement/instrumentation/control/systems engineers. Thus, we adapted the KAOS method to fit the different special features of process control systems, such as being component-based, the nature of goal refinement patterns, the relationships between the components, and non-termination of reactive systems (a continuous loop of sensing environment variables and reacting to changes in them). Moreover, we augmented the checking phase by adding completeness, reachability and animation utilities; in addition, and of equal importance to the requirements method enhancements, the tool automatically generates a B formal specification.

This paper consists of four sections; in section one, we introduce the research areas of requirements engineering, formal methods, and the goal-oriented method of KAOS. In section two, we briefly describe the GOPCSD development process and the provided requirements library. In section three, we examine a production cell case study to explain the use of the GOPCSD tool. Finally, in section four, we compare our method with related methods and suggest directions for future work.

2 The GOPCSD method and its supporting tool

The GOPCSD tool [5] has been developed to decrease 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 [6], 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.

2.1 Requirements elements within the GOPCSD tool

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

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, but not within the GOPCSD tool integrated development environment in order to ensure consistency of components across applications.

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's user can still create, edit, and delete variables from the application design space.

The Agents. Agents are the objects that control the application 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 pre-defined tasks (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 when the user imports 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.

The Goal-models. Goal-models constitute the main segments of the structured requirements; they represent the user requirements in 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 safety, security quality or operational requirement; the tool supports informal descriptions of goals as well as formal descriptions based on temporal logic [15]. In the GOPCSD tool, goal-models can be used as workspaces until the user manages to construct one complete goal-model that specifies the whole of the application requirements and each of the terminal goals, of this goal-model, has been assigned to an agent.

2.2 Goal Refinement Patterns

The main task of the systems engineer is to build satisfactory goal models of requirements. Identifying goal refinement patterns closer to the systems engineer's perspective should have a strong impact on reducing the effort required to construct the goal-oriented requirements, as well as the ease of understanding the requirements model by the systems engineer. Thus, we examined different process control systems and identified the following useful patterns:

The Alternative Refinement Pattern. The alternative refinement pattern is the KAOS [20] OR pattern. This pattern means that each of the sub-goals can be used independently to achieve the main goal. Accordingly, the alternative pattern should be used when there is a preference to express the availability of different solutions within a single goal-model. For example, to achieve the goal for shutting down a gas burner system, and taking into account the safety considerations, the 'shut down' goal can be refined as an alterative of two sub-goals: the first goal - the gas valve will be closed first and then the air valve - and the second goal - the two valves will be closed simultaneously. At a later development stage, each of these two sub-goals will appear in a separate final version of the goal-model.

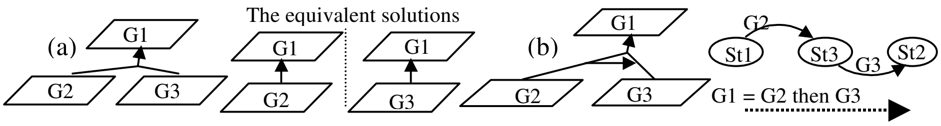

Fig. 1, *(a) the alternative pattern and (b) the sequence pattern*

In fig. 1a, we sketch a goal-model template to the left, where the non-terminal goal G1 is refined as the alternative of the two goals G2 and G3. To the right, a set of two goal-models is generated as a result of splitting the goal-model to the left at the alternative refinement site. Each sub-goal G2 and G3 appears separately in one of the two goal-models and the refinement relationship in the resulting goal-model will be *Inheritance* instead of *Alternative*.

The Sequence Refinement Pattern. The sequence pattern is frequently used in process control systems. It is used to express a higher-level task in terms of a number of ordered steps; when each of these steps is achieved in order, the higher level task will be achieved.

For example, a goal of delivering processed metals in a production cell application can be refined as a sequence of three goals: the first goal of feeding the blank metal to the blank processor, the second goal of processing the blank metal, and finally the third goal of delivering the processed blanks. When a goal is refined to a sequence of sub-goals, each sub-goal's accomplishment is considered as a pre-condition for its successor goal.

In fig. 1b, the goal-model template to the left expresses that goal G1 is refined as the sequence of goals G2 and G3. There is an equivalent state transition diagram (STD) corresponding to the goal-model. Each goal in the goal-model is represented as a transition in the STD; the states of the STD correspond to the pre- and post-conditions of each goal. This analogy shows that the intermediate state of the STD (*St3)* represents the pre-condition of goal G3 and at the same time represents the post-condition of goal G2. Therefore, as noted above, within the sequence refinement pattern, the post-conditions of the predecessor goals must be considered.

The Conjunction Refinement Pattern. Conjunction patterns are usually used to refine the main goal into two or more sub-goals that need to be conjunctively accomplished. If the sub-goals do not possess any sequence or order dependency between each other, we can express the sub-goals as conjunction refinement of the parent goal. The main goal will be considered as achieved only when each of its sub-goals is achieved. This refinement pattern is more likely to exist at the high-level segments of the goal-model where the sub-goals themselves are non-terminal and need further refinements. For example, one should consider the different process control requirements aspects like safety, running cost, security, and operation. These aspects are usually represented as high-level goals that can be refined to narrower sub-goals. Thus, the issue of bringing together these aspects can be represented as a conjunction between the high-level goals.

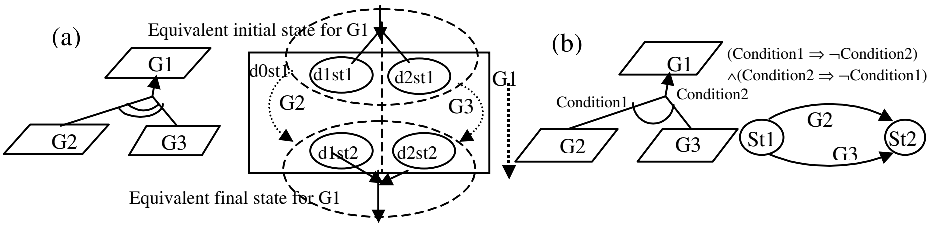

Fig. 2, *(a) the conjunction pattern and (b) the disjunction pattern*

In fig. 2a, goal G1 is refined as the conjunction of goals G2 and G3. To the right hand side, there is a sketch of an equivalent STD that has two sub-spaces, one for each conjunct sub-goal. This compound STD illustrates that the pre-conditions of goals G2 and G3 should together imply the pre-condition of goal G1. In addition, having achieved the post-condition of goals G2 and G3 should guarantee the achievement of the post-condition of goal G1.

The Disjunction Refinement Pattern. This pattern has some features of the KAOS OR and AND patterns. It resembles the alternative pattern in the sense that the parent goal can be achieved by achieving one of the sub-goals. On the other hand, the different sub-goals achieve the parent goal in different situations; thus, removing one of them means that the parent goal will not be achieved in some circumstance. This pattern is repeatedly found in process control systems that achieve tasks differently from one situation to another.

For example, a double press production cell can stamp the blank metals in one of the presses according to availability of the presses; thus, a parent goal to stamp metals can be refined to two sub-goals: the first is to stamp the metal in press one under some condition and the second goal is to stamp the metal in press two under some other condition, but guaranteeing that the first condition has not been met. This indicates that the two sub-goals are disjoint.

In fig. 2b, goal G1 is refined as the disjunction of goals G2 and G3. To the right there is a sketch of a STD that has two transitions, one for each sub-goal. Furthermore, we can notice that the pre-condition of G2 added to -logical or - the pre-condition of goal G3 implies the pre-condition of goal G1. Or we can state that goal G1 is active when any of goals G3 and G2 is active. The post-condition of goals G2, G3 and G1 should be the same.

The Simultaneous Refinement Pattern. The simultaneous pattern is commonly required within process control systems. The parent goal will be refined to a number of simultaneously operating goals that may control different parts of the application.

For instance, in chemical applications, some situations can arise where pouring two chemical compounds or adding a substance and stirring the mixture are required to commence at the same time.

Although to some extent the simultaneous pattern is similar to the conjunction pattern, they differ in the restriction for starting the actions. This difference delays the appearance of the simultaneous pattern to one level above the terminal-goal level, where the sub-goals of the simultaneous pattern will be terminal goals. This powerfully employs the simultaneous pattern to synchronize the operation of sub-components of the application.

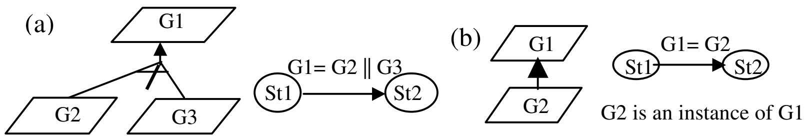

Fig. 3, *(a) the Simultaneous pattern and (b) the inheritance pattern*

In fig. 3a, goal G1 is defined as the simultaneous refinement of goals G2 and G3. The equivalent STD to the right of the goal-model shows that the post-condition of goal G2 added to, logically $\wedge$, the post-condition of goal G3 implies the post-condition of goal G1. Moreover, the pre-conditions of G2 or G3 should imply the pre-condition of goal G1.

The Inheritance Refinement Pattern. The inheritance pattern is a special case of refinement patterns where there is only one single sub-goal[1] that has the same formal description as the main goal.

There are two main motivations for defining this pattern: first, when the main goal is a soft goal and too general to be refined directly, there will be a need for an intermediate goal that is stronger/harder and/or more specific than the main goal. For

[1] To avoid any circumstance of ambiguity and inconsistency

example, in a lift system, defining a goal like maximizing the lifetime of the cabinet motor would be too general to be refined. But, a goal like avoiding unwanted direction change could bridge the big distance between the level of motor and requests, on the one hand, and the high-level issue of motor lifetime, on the other. The second motivation is to maintain the refinement level after splitting the compound goal-models, where inheritance sites will replace the alternative sites in the new simple goal models. This replacement fills the large refinement distance between goals G2 or G3, on the one hand, and the parent goal G1, on the other.

In fig. 3b, goal G2 inherits goal G1; to allow the child goal G2 to be stronger than the parent goal (i.e. a more restricted pre-condition), the pre-condition of the child goal G2 should imply the pre-condition of the parent goal G1; while the post-condition of the parent should imply the post-condition of the child goal.

2.3 The GOPCSD Library

To reduce the effort required to construct a process control system as well as to shorten development time, a library that stores the specification of the individual components is required. The elements stored in the library should represent the most frequently occurring requirements, which we model as the low-level components and the application's high-level goal templates. Importing the components and the templates from such a library enables the user to solely focus on integrating the separate components and specifying the system's medium level goals, to connect the high-level templates and refine them into low-level components' goals.

3 Analysing the requirements of a Production Cell

To present the method and assist in illustrating the tool, we examine a case study of a simple production cell [14], a recurring example in the literature. The production cell receives blank metals, stamps them and finally, delivers the processed ones to the next stage or the collection area. To accomplish this, the production cell consists of a feed-belt, rotary table, robot, press, and a deposit belt, as shown in figure 4a. This system has different requirements aspects as stated in [14], such as, safety, liveness, operational and increasing the throughput. Moreover, non-behavioural requirements are also considered as important, such as being easily extended, understandable and modifiable.

The process of production cell development assumes having the individual component descriptions in the GOPCSD library, as well as general templates to outline the requirements hierarchy. (These components and templates can be stored before developing the production cell by using the GOPCSD library manager program. The purpose of having these in the library is that production cells are much of a likeness and so these elements can often be reused.) The development process allows the user to start with an imperfect model and then attempt to guide him/her in debugging and enhancing these initial models; then, the tool automatically generates a B specification and/or general-format formal operation definitions. In the following sections, we illustrate the development process in some detail.

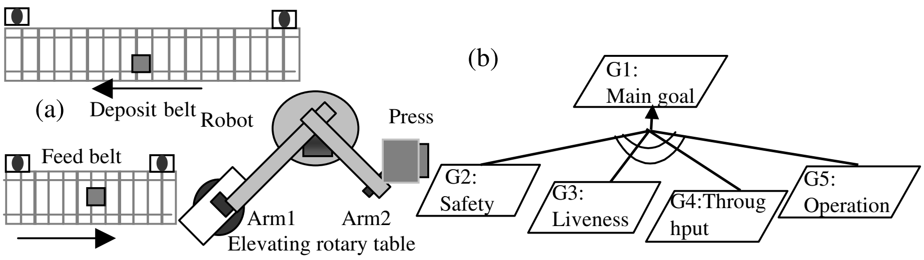

Fig. 4, the production cell: (a) the component assembly and (b) the abstract requirements

3.1 Constructing the Production Cell Requirements

In the construction phase, the systems engineer can best start the development process by importing the production cell components and templates offered by the GOPCSD library. The components' low-level goal-models have the detailed specifications of the operation of the robot, rotary table, press, deposit belt and feed belt. In addition, high-level production cell templates such as *feed-metals* and *deliver-metals* serve as outlines for the user to shorten the requirements development time, as well as guiding the structuring of the requirements.

If the templates' sub-goals do not cover all of the user needs, he/she can create goals to trace these missed needs and combine them with the existing goal-models. Having imported the components and templates and defined the extra user related goals, the user should now focus on refining the incomplete goal-models and combining the separate goal-models together.

The different requirements aspects of the production cell should appear directly under the main goal of the production cell. This can be achieved by refining the main-goal G1 into safety G2, liveness G3, throughput G4, and operational G5 goals, as shown in figure 4b; thus, this goal can be refined as a conjunction of four sub-goals, each of them addressing one of these aspects.

After this step, the incomplete goals should be refined using the previously defined patterns; the refinement process should be continued until each goal is simple enough[2] to be assigned to one of the application agents.

Safety. The safety goal G2 can be refined as a number of conjunctive goals that are designed to avoid the machines colliding with each other (G6), avoid hitting any objects by the robot arms (G7), avoid having two metals on the table tray at the same time (G8) or inside the press at the same time (G9). Goal G6, avoiding collision between the different machines can be refined as -goal G10- avoiding the table hitting robot arm1, and –goal G11- avoiding the press hitting robot arm1 and –goal G12- arm2. This refinement is shown in fig. 5.

[2] Each output variable controlled by the goal can be manipulated by a single agent

Liveness. Goal G3, the liveness goal, ensures that each blank metal coming into the production cell will eventually be processed and delivered to the next processing stage or the collection area. This can be achieved by refining the goal into sub-goals, each concerned with liveness at a specific stage of processing; thus the goal can be refined as the conjunction refinement of avoiding dropping the blank metals from the feed metals G13, avoiding pushing of the metals out by the robot G14, avoiding the robot picking up unprocessed metals G15, and avoiding dropping the metals G16; this can be refined as: avoid the robot dropping the blank metals anywhere except inside the press G17 and avoiding the robot dropping processed metals anywhere except at the start of the deposit belt G18.

Throughput. Goal G4: the throughput goal can be construed in terms of reducing the service time for each metal; this can be decomposed into sub-goals, each of which attempts to reduce the local service time at each different stage. Therefore, a possible refinement could be as follows: G19 maintain a continuous stream of incoming blank metals, G20 keep the robot ready to serve either the press or the table, G21 keep the table ready to receive new blank metals, G22 keep the press ready to process a new metal, and G23 keep delivering the processed metals.

Operation. Goal G5 concerns the normal operation of the cell. The operation of the cell can be abstractly refined as a pipelining of feeding blank metals, processing them, and delivering the processed ones. This suggests refining goal G5 as a conjunction of three sub-goals; these three sub-goals do not require further effort because the GOPCSD library already offers them: the feed-metals and deliver metals templates and the process-metals as one low-level compound goal-model of the press Component.

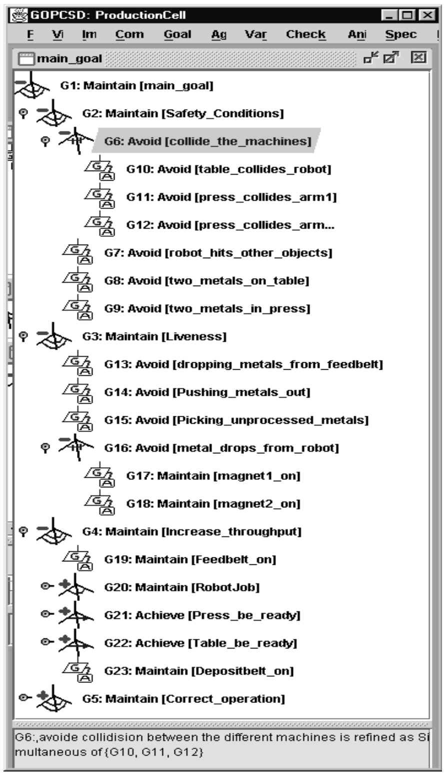

Fig. 5, the goal-model of the production cell after refining goals G2, G3 and G4.

3.2 Checking the Requirements

After constructing the goal-model of the production cell, the tool should guide the user to capture the requirements bugs that may still exist within the specified requirements. The checks and tests, which the requirements undergo in this phase, serve as bottom-up feed back; this guides the user to obtain more realistic requirements, as well as allowing the user to enhance the requirements after observing the animation. In the following sections we list the formal and informal checks that can be applied to the requirements within the GOPCSD development environment.

Checking the refinement constraints. Before checking the formal parts of the goal-model, the user can debug the goal-model by checking the correctness of the goal-model and whether each functional terminal goal has been assigned to an agent or it is considered as a non-functional goal. Also, the tool ensures that each output variable accessed by the terminal goal is uniquely controlled by the same agent. Moreover, the tool checks whether the refinement constraints (that we have mentioned in section 2) are met.

Reasoning How and Why about goals. As a first check, the user can reason about the why or the how for any of the goals; this reasoning can motivate the user to elicit new goals and/or modify existing ones by tracing the informal descriptions of the sub- and super goals.

Obstacle Analysis. Obstacle analysis as described in [21] is very useful to perform, especially in medium- or large-scale systems. Obstacle analysis requires further attention from the user, in considering conditions not dealt with in the goal model that can inhibit one of the goals from being accomplished.

For example, in goal-model *main-goal*, goal G19, which maintains the feed belt motor switched on to increase the throughput, will be obstructed when the rotary table is not ready to receive new blank metals or when it is facing the feed belt and is at the same level as the feed belt but already has a metal. In these two cases, the motor cannot be kept on otherwise metals will drop from the belt or two metals will be placed on the table, neither of which is a safe condition.

These two conditions can be considered as two obstacles that occur when the pre-condition of the obstructed goal is satisfied and the negation of its post-condition is negated, i.e. *feedbelt_motor = OFF*. The GOPCSD tool enables the user to report such cases and suggest different ways of resolving the obstacle according to its type. Having performed the obstacle analysis and modified the goal-model, the requirements will be more realistic, taking greater cogniance of the local environment where the system is supposed to operate.

Goal-Conflict Analysis. Goal-conflict analysis [4, 22] enables the user to remove any inconsistency that might exist within the requirements. The goal-conflict arises when two or more goals prescribe the performance of inconsistent actions at the same time. The GOPCSD tool enables the user to perform goal-conflict checks.

After ensuring the user provided a valid goal-model, it is important to ensure that the goals of the goal-model are consistent. Otherwise, possible deviations and unexpected scenarios may take place during run-time. Goal conflicts are checked by comparing the conditions for goals that assign different values to the same variables.

Resolving the goal-conflicts will guide the user to bring together the safety, operational, liveness and throughput requirements aspects. For example, a conflict case was reported between goals G10 and G37: goal G10 (safety goal) attempts to stop the table from rotation when the robot arm is still extended, whereas goal G37 (throughput goal) attempts to return the table as quickly as possible to its normal position to receive a new blank metal. Thus, the conflict can be resolved by strengthening goal G37's pre-condition to consider the safety condition.

Reachability Check. Usually, in large-scale systems there is a possibility of having errors within the goal's formal description, which can result in useless goals that will never become active. The GOPCSD tool can detect such cases provided that the application variables have finite domains. The pre-condition of the unreachable goal may contradict one of its ancestor goals' pre-conditions or the predecessor goal's post-condition (in case of sequence patterns). By reporting these unreachable goals to the user, he should be able to correct the formal description.

Completeness Check. Completeness analysis can reveal situations, which are not considered by the systems engineer. In the GOPCSD tool, we follow the definition of completeness in [3]. A completeness check verifies the condition that for each combination of the application variables there should be a defined action(s) to be taken, and these actions determine the output variables.

Although for some variable combinations the user may decide that these situations will not happen, he/she still needs to be informed of these situations. For example, having checked the completeness of the goal-model *main-goal*, we detected the following case: when *Press_press_state = OFF and Press_has_metal = YES and Press_level = UP and Press_metal_pressed = NO and Robot_angle = ARM1_TABLE and Robot_arm1_length = RETRACTED and Robot_arm2_length = RETRACTED*, the output variable *Press_v_moving* is not fully determined. In fact, this case reveals that when the press is pressing a blank metal, the press tray vertical motion is not fully determined; this may require from the user the addition of a requirement stating that the motor must stop until the pressing operation finishes.

3.3 Validating and Modifying the Requirements

After the tool guided the user to discover and remove most of the requirements' bugs, the user should revise the system operation and observe its performance using validation utilities. This is required to obtain his/her agreement to the gradually stated and formalized requirements. In addition, this can be considered as a late checking phase to correct or enhance any particular requirement.

As shown in fig. 6, the user can observe the activated goals cycle by cycle. Further more, different stimuli can be applied to the system and the system response can be observed and saved, for further analysis. Moreover, the tool enables the user to save and load and execute lists of stimuli, as well as simulating delays in output variables and/or faults.

One of the basic concepts of the GOPCSD is to guide the systems engineer to correct/complete/enhance the requirements rather than starting with correct and complete ones. To achieve this, the tool provides different feedback approaches to help the user modify or add goals of the checked goal-model. Although these checks and tests attempt to enhance the requirements from different points of view, they all share a feedback guidance based on goals recommended for change or modification and actually recommends modifications to perform. These guided modifications enable the systems engineer to enhance the requirements; in addition, they reduce the dependency of the final accepted requirements on the initial imperfect interpretation of user needs.

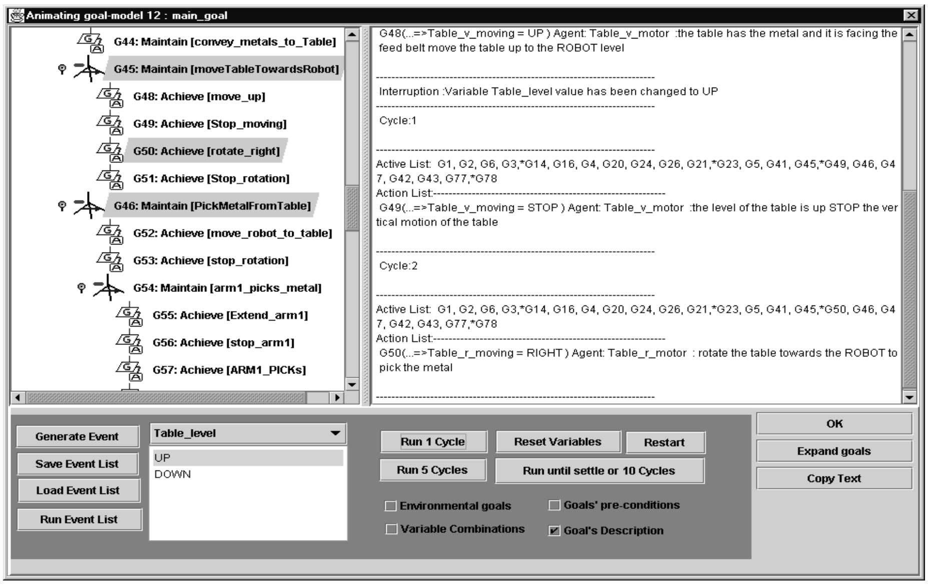

Fig. 6, validating the requirements using the animation utility offered by the GOPCSD tool

3.4 Generating Formal Specifications

The third phase, for generating the B specifications, will be hidden from the systems engineer; thus he/she does not have to know much about B or formal methods. However, the GOPCSD tool produces documented formal specifications, to the extent that this is possible, using the informal descriptions of the goals, variables, agents and data types.

Generating Formal operations. The general operations specifying the Production Cell will be generated from the terminal goals of the *main_goal* goal-model. The tool generates operations with pre- and post-conditions and an actor or agent who is capable of performing such operations. Each operational terminal goal (which is neither an environmental nor a non-functional goal) will be translated into an invariant. The generated operations for the production cell are listed in table 1.

Table 1. Sample of the Production Cell operations

Operation no. 1 table_collides_robot	Actor(Agent): Table_r_motor
/* G10: table_collides_robot avoid table collides robot*/ **Pre-condition:** Table_level = UP and Table_angle = ROBOT_FACING and Robot_arm1_length = EXTENDED **Post-condition:** avoid table collides robot Table_r_moving = STOP	
Operation no. 2 press_collides_arm1	Actor(Agent): Press_motor

/* G11: press_collides_arm1 avoid the press tray hitting the robot arm1 when it is in the middle position*/
Pre-condition: Robot_angle = ARM1_PRESS and Robot_arm1_length = EXTENDED and Press_level = MIDDLE and not (Press_v_moving = STOP)
Post-condition: avoid the press tray hitting the robot first arm when it is in the middle position
Press_v_moving = STOP

Generating B machines for the Production Cell. The tool generates the B machines guided by the variables' relationships within the goal-model. In the case that some output variable does not appear in the goal model, it will not be present in the specification. The tool generates the following machines and associated comments from the requirements information data:

Data types machine. A machine involving the definition of the variables' types; this information is collected from the variables' details created within the application or from the library. The following B segment is a sample from the data type machine:

```
MACHINE Datatypes
SETS
ONOFF := {OFF, ON }    /* two values motors */;
VERTICAL_MOTION := {UP, STOP, DOWN } /*motors that move up/down/stop */;
```

Actuator machines. These machines representing the actuators, based on the different operations that each agent could perform, as defined within the terminal goals. The following B segment is a sample related to the angular motion of the rotary table actuator machine:

```
MACHINE Table_r_moving_actuator
SEES datatypes
VARIABLES
Table_r_moving /* output variable: the direction of the rotation */
:
```

A main controller machine. This machine is based on the goal-model hierarchy, representing the terminal goals as pre- and post conditions; the informal details of the terminal goals will be generated as comment lines at the proper sites within the B code to enable the software engineer to understand the operation of the application. The following B segment is a sample from the main controller machine:

```
:
/* G48 move_up:  the table has the metal and it is facing the feed belt
move the table up to the ROBOT level */
 ( Table_angle = FEED_BELT_FACING & Table_level = DOWN & not (
Table_v_moving = UP )) & Table_has_metal = YES
THEN
set_Table_v_moving_UP /* an operation of an actuator machine */
ELSIF...
```

These B machines can be further refined and processed by the software engineer within the B toolkit environment with a high confidence that the systems engineer has agreed the requirements, until executable code is eventually generated.

3.5 Extending the Production Cell

The non-behavioural aspects of requirements specifications, such as being easy to modify and easy to evolve, as indicated in [3], can be assessed by considering the effort required to extend the production cell application. We examine two cases as follows:

Double-Press Production Cell. To increase the throughput of the production cell, another press can be added. When a metal needs to be pressed, the robot can select one of the presses, if both are free, or place it in the free one. In this case, the requirements goal-model developed for the simple production cell can still be used. The goal-model's segments, which do not address the press can be copied as is, whereas those segments addressing placing a metal in the press require further effort (prepare a copy from the original goal for each press and then combine these two copies using the disjunction pattern to form a higher goal to press the metal).

Cascaded Production Cell. To process the metals in two cascaded stages, we can build a higher system composed of two simple production cells (each of them considered as a simple component). The first cell's deposit belt and feed belt of the second cell need to be mapped to the same physical component; this facility is already offered by the GOPCSD tool. The cascaded production cell's main goal can be simply constructed by conjunctively combining the main goals of each simple production cell.

4 Conclusions

Having noted the gap between the systems engineer's perspective and the formal specification level familiar to the software engineer, we were motivated to start the requirements analysis as close as possible to the view of the systems engineer, as in [12], and, furthermore, extend the formalisation, via an automated tool, to the B specification level, as in [10, 18].

The GOPCSD tool implements the early requirements stages. It enables the user to improve and enhance the initial requirements model by removing the requirements bugs as early as possible; it decreases the interference of concerns between the systems and software engineers. Moreover, it aims at not only providing user guidance through moving down from the abstract user needs to the early specifications, but also to effectively conduct the up correcting movement from specification level to the user needs. This results in acquiring realistic, enhanced and user-agreed requirements based on the feedback of the various checks and tests offered in the GOPCSD tool.

In relation to generating B specifications for reactive systems, the GOPCSD tool differs from [10, 18], where the user needs are not clearly traced to the generated specification level. The concept of building systems from components/sub-systems, as in [13], is not supported by a library in [9], as in the case of the GOPCSD tool. Also, having alternative goal refinement pattern, the GOPCSD tool is able to develop and check different versions of the control program. Due to its goal-oriented nature, the GOPCSD supports informal reasoning about goals and grouping the related specifications together; which is not the case in [7, 8, 9, 10, 18].

Compared to the KAOS method [20], the GOPCSD tool has attained stronger attributes by narrowing its scope to the process control family. This enabled us to identify effective refinements pattern, as in [5], to guide the systems engineer developing and understanding the applications requirements. Instead of generating state transition machines as in [12], the GOPCDS tool generates B machines as the main output, which is thought to be closer to a client's expectation, as it is generated directly from a formalised and validated requirements specification, and can be further processed by software engineer within the B toolkit [2] or other similar environments to generate code that is also likely to be closer to client expectations. The GOPCSD tool offers built-in animation utilities, which hide the details of state transition machines from its user and provide direct feedback and visual highlighting of the activated goals cycle by cycle.

Accordingly, we recommend that further research can focus on a sub-family of Process Control Systems, like chemical systems or automotive applications, to increase the chance of identifying specific and effective refinement patterns and building a library of more specialized low-level components. Also, some attention should be given to supporting the maintenance stage of existing applications, where the abstract requirements could be extracted as goal-models from the existing formal specifications. Finally, generalizing fault-tolerance and HAZOP analyses [19], to represent them as high-level templates within the GOPCSD library, should reduce the effort required by the systems engineer to carry out separate analyses for risk and safety.

References

1 J. R. Abrial, "The B Book: Assigning Programs to Meaning", Cambridge University Press, 1995.

2 B Core UK limited (1998), B Toolkit http://www.b-core.com/btoolkit.html

3 Alan M. Davis, Software Requirements: Objects, Functions and States, Prentice Hall PTR, 1993

4 S. Easterbrook, "Resolving Requirement Conflicts with Computer-Supported Negotiation", In requirement engineering: social and technical issues, M. Jirotka and J. Goguen (Eds.) Academic Press, 1994, pp 41-65.

5 I. A. El-Maddah and T. S. Maibaum, goal-oriented requirements analysis of process control systems, 1st MemoCode, France, 2003

6 I. A. El-Maddah and T. S. Maibaum, Goal-oriented Process Control System Design Tool, Tool Exhibition of FM03, Italy, 2003

7 Constance L. Heitmeyer and John McLean. Abstract requirements specifications: A new approach and its application. IEEE Trans. Software Eng., SE-9(5):580-589, Sept. 1983

8 M. Jaffe et al "Software requirements analysis for real-time Process Control Systems", IEEE transactions on software engineering 17,3 (March 1991); 241-58

9 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

10 K. Lano, K. Androutsopoulos, and D. Clark, Structuring and Design of Reactive Systems using RSDS and B, FASE, ETAPS 2000

11 Gary T. Leavens and Murali Sitaraman, Foundation of Components based Systems, Cambridge Press, 2000

12 E. Letier and A. van Lamsweerde, Deriving Operational Software Specifications from System Goals, SIGSOFT 2002/FSE-10, Nov. 18-22 Charleston, SC, USA.

13 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).

14 Claus Lewerentz and Thomas Lindner, Case Study "Production Cell" a comparative study in formal software development, FZI Karlsruhe, 1995

15 Z. Manna and A. Pnueli, "The Temporal Logic of Reactive and Concurrent systems", Springer-Verlag, 1992.

16 S. Mellor and P. Ward, Structured Development for Real-time Systems (3 volumes), Yourdon Press, 1985.

17 Bashar Nuseibeh and Steve Easterbrook, Requirements Engineering: a Roadmap, Future of Software Engineering, Limerick , Ireland, 2000

18 E. Sekerinski, Graphical Design of Reactive Systems, D. Bert (Ed.) 2nd International B conference, Montpellier, France, Springer-Verlag, 1998.

19 Neil Storey, Safety Critical computer systems, Addison-Wesley Longman, 1996

20 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.

21 A. Van Lamsweerde and E. Letier, "Obstacles in Goal-driven Requirement Engineering", proceeding ICSE'98 20th international conference on software engineering, Kyoto, ACM-IEEE, April 1998.

22 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.

23 P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering", ACM Trans. Software Eng. And Methodology, 6(1). 1997. 85.