A FORMAL APPROACH TO DEPLOY HETEROGENEOUS SOFTWARE COMPONENTS IN A PLC

Mohamed Khalgui and Emanuele Carpanzano
ITIA - Institute of Industrial Technologies and Automation CNR 20131 Milan, Italy

Keywords: Component Based Technologies, Industrial Control Systems, Multi-tasking PLCs, Deployment, Real-Time Scheduling.

Abstract: This paper deals with an industrial control application following different component-based technologies. This application, considered as a network of heterogeneous components, has to be deployed in a multi-tasking PLC. It has classically to respect temporal constraints according to specifications. To deploy the components in feasible OS tasks of the controller, we propose to fix a formal component model allowing their homogeneous design. We enrich, in particular, this model to unify well known technologies. The application is considered then as a network of homogeneous components. We propose to transform this network into a real-time tasks system with precedence constraints to exploit previous results on real-time deployment.

1 INTRODUCTION

The development of critical industrial control applications is nowadays a very complicated activity basically due to the always increasing set of functional and non-functional requirements. Control applications have to satisfy stringent real-time constraints that are difficult to be addressed following the traditional development approaches. This problem is more complicated by the continuously increasing demand for shorter development time. This also imposes the demand for shorter verification phase. The component based development is widely accepted by industry as a successful paradigm to address these requirements (Crnkovic and Larsson, 2002).

Nowadays, several component based technologies have been proposed to develop control applications (Crnkovic and Larsson, 2002). These technologies allow to reuse already developed components available in rich libraries. In addition, they support the modularity reducing the design complexity. Nevertheless, the majority of these technologies depend on particular companies or projects (Crnkovic and Larsson, 2002). Therefore, the use of the corresponding libraires is limited.

In this paper, we propose to reduce the realization time of applications by exploiting the advantage and also the component library of each one of the known component-based technologies. Therefore, a control application is a network of heterogeneous components following different technologies. These components have classically to respect functional and temporal constraints described in specifications. To our knowledge, there is no approach considering such functional architecture of component-based applications. The problem that we tackle in this paper is how can we validate the application components whereas they are developed using different technologies (figure 1)? Moreover, how can we deploy these components in feasible OS tasks of a PLC ?

To resolve this problem, we propose to fix a formal component model (Sifakis, 2005) to unify all the known component-based technologies. In addition, we enrich this model to take into account characteristics of these technologies. Thanks to this model, the application is transformed then into a network of homogeneous components having the same characteristics. To correctly deploy these components, we propose to transform them into a real-time tasks system with precedence constraints. The purpose is to exploit previous results on deployment and scheduling of component-based applications (Khalgui et al.,

Khalgui M. and Carpanzano E. (2007).
A FORMAL APPROACH TO DEPLOY HETEROGENEOUS SOFTWARE COMPONENTS IN A PLC.
In *Proceedings of the Second International Conference on Software and Data Technologies - SE*, pages 207-212
DOI: 10.5220/0001340502070212

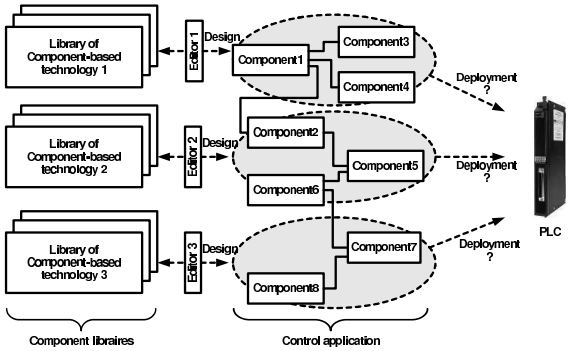

Figure 1: Deployment of a control application based on heterogeneous components.

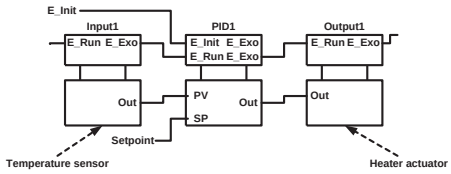

Figure 2: A FB Component : *TemperatureRegulator*.

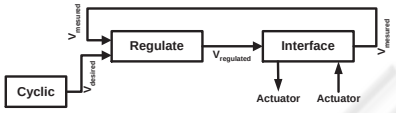

Figure 3: A PBO Component : *Speed Regulator*.

2006).

In the section 2, we briefly present the well known industrial component-based technologies. Then, we detail a particular approach that we exploit in all the continuation as a unifying component model. In the section 4, we propose to enrich this formal approach to be compliant with the known industrial technologies. To deploy a control application, we propose in the section 5 to transform the corresponding homogeneous components into a subtasks system with precedence constraints.

2 COMPONENT-BASED TECHNOLOGIES

Nowadays, several component-based technologies have been proposed to develop industrial control applications (Crnkovic and Larsson, 2002). These technologies depend often on particular industrial companies or projects. Each one proposes a particular characterization of the component concept. In this section, we briefly present the most known technologies in industry.

The IEC 61499 standard (Crnkovic and Larsson, 2002) is a component-based standard allowing the development of distributed control applications. According to this standard, a function block is a an event triggered component owning data and an application is a network of blocks. In the figure 2, we present a simple example of a *Temperature Regulator* proposed in (Lewis, 2002). This regulator is composed of two interfaces blocks (*Input*1 and *Output*1) and a regulation block *PID*1. A detailed description of this example is available in (Lewis, 2002).

The Carnegie Mellon university proposes also its owner component concept named Port Based Object (Crnkovic and Larsson, 2002). This technology is often used to develop industrial control applications in robotics. A PBO component is a particular case of a Function Block. The figure 3 presents the *Speed Regulator* as a simple example of PBO components regulating the vehicle speed. The component *Cyclic* sends periodically the desired values to *Regulate* which regulates the values measured from the component *Interface* (Crnkovic and Larsson, 2002).

The Arcticus Systems propose also another component model called "Rubus" (Crnkovic and Larsson, 2002). This technology allows to consider functional and temporal constraints on application components. In the figure 4, we present *BrakeSystem* as a simple example of Rubus components to use in a vehicle. The component *BrakeLeftRight* allows to brake left or right by considering the pressure and also the speed of the vehicle.

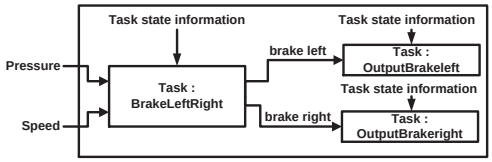

Figure 4: A Rubus Component : *BrakeSystem*.

By studying these component-based technologies, the component concept is quietly the same from a technology to another. In addition, these technologies provide rich libraries. Therefore, it is interesting to exploit them in the order to reduce the realization time of an industrial control application. In all the continuation, we consider a control application as a network of heterogeneous components following different technologies.

3 FORMALIZATION

To cover all the component-based technologies, we propose to fix a formal approach unifying the component concept. Nowadays, several formal approaches have been proposed. In this paper, we select the best one that supports a detail specification of interactions between components (Sifakis, 2005). This approach defines classically a component as interfaces and an implementation. Let us consider an application containing K components. According to (Sifakis, 2005), the application is characterized by a set of actions A supporting its different functionalities. An application component $comp_i$ $i \in [1,K]$ is a subset of actions A_i ($A_i \subset A$). An action is implemented by an algorithms sequence in the component. We note, in addition, that the application components are with disjoined subsets of actions ($A_i \cap A_j = \emptyset$). To specify an application, (Sifakis, 2005) proposes the following models :

- **Interaction model** : defines the interactions between the application components.
- **Behavior model** : defines the behavior of each application component.
- **Execution model** : defines a fixed priority policy allowing the execution of the application components.

In this paper, we just describe the two first models.

3.0.1 Interaction Model

To specify interactions between components, (Sifakis, 2005) defines the *connector* concept. A connector c defines a maximally compatible set of interacting actions between components. It is a non empty subset of A such as

$$\forall i \in K, |A_i \cap c| \leq 1$$

Given a connector c, an interaction α of c is defined in (Sifakis, 2005) as any term of the form

$$\alpha = a_1|a_2....|a_n, \{a_1,....,a_n\} \subseteq c$$

The operator "|" is a binary associative and commutative operator. It is used to denote a composition of actions. The interaction $a_1|a_2....|a_n$ is the result of simultaneous occurrences (or execution) of the actions $a_1,.....,a_n$. Note that if $\alpha = a_1|a_2....|a_n$ is an interaction of a connector, then any term corresponding to a subset of $\{a_1,a_2....,a_n\}$ is also an interaction. Finally, one denotes by $I(c)$ (resp, $I(C)$) the set of interactions corresponding to a connector c (a set C of connectors).

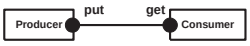

Figure 5: Producer / Consumer composition.

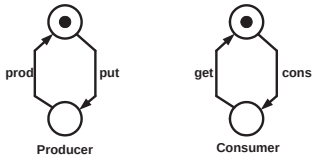

Figure 6: The behavioral model.

3.0.2 Behavioral Model

According to (Sifakis, 2005), the behavior of an application component is defined as a transitions system. This system is classically characterized by a triple $(Q, I(A), \rightarrow)$ where,

- Q is a set of states,
- $I(A)$ is a set of interactions between actions of A,
- $\rightarrow \subseteq QXI(A)XQ$ is a transition relation. As usual, one writes $q_1 \xrightarrow{\alpha} q_2$ to denote a transition between q_1 and q_2.

3.1 Example

To illustrate these two models, we present in the figure 5 the classic example of Producer/Consumer. These two components interact together through the $put|get$ of the connector $\{put, get\}$. We present in the figure 6 the behavioral models of the two components. We note that *prod* and *cons* are internal actions of the components *Producer* and *Consumer*.

4 EXTENSIONS AND ASSUMPTIONS

Although this formal approach is expressive to model component based applications, it is not compliant with the known component-based technologies. Therefore, we propose to enrich this approach by several concepts. We propose to classify first of all the component actions in input, internal and output actions. In addition, we propose a functional and temporal characterization of a control application.

4.1 Component Actions

Let c be a component of a control application. We propose the following sets to characterize the corre-

sponding actions :

- $external(c)$: the set of input/output actions (in c) interacting with outside.
- $internal(c)$: the set of internal actions (in c). An internal action can be activated by more than one input action.

We define for each internal action *act* of a component c ($act \in internal(c)$) the following external actions :

- $input(act)$: a set of input actions activating *act* ($input(act) \subseteq external(c)$).
- $output(act)$: a set of output actions to activate once the execution of *act* finishes ($output(act) \subseteq external(c)$).

Finally, we denote by *In_Act* (resp, *Out_Act*) the set of input (resp, output) actions in the application ($\forall act \in internal(c)$, $input(act) \subset In\_Act$, $output(act) \subset Out\_Act$).

4.2 Composition of Components

To be compliant with several industrial component-based technologies, we enrich the functional architecture of a control application.

4.2.1 Container Concept

In several component-based technologies (Crnkovic and Larsson, 2002), the container concept is proposed to gather application components controlling physical processes. A container is a logical execution unit corresponding to time slots of the processing unit. In this paper, we propose the following definition of a container.

Definition. *a container is a set of application components sharing the control of physical processes. It is characterized by a sequencing function defining the static scheduling of the internal components. We apply a non-preemptive policy to process this function.*

In the operational architecture, we propose to consider a container as an OS task. This task implements all the possible execution scenarios of the internal components. The sequencing function of the container is the "main()" function of this task.

4.2.2 Temporal Constraints

We propose the function *cause* specifying causalities between output actions of application components and input actions of another ones. Two actions are under a causality constraint if they belong to a same interaction of a connector.

$$\forall ia \in In\_Act,$$

$$cause(ia,c) = \{\varphi \in Out\_Act / \exists i \in I(C), \{ia\} \cup \varphi \in i\}$$

In the Producer/Consumer example, we note that $cause(get, consumer) = \{put\}$. According to specifications, we define End to End Response Time Bounds between periodic readings from sensors and the activation of the corresponding actuators. The scheduling of the application components has to take into account these bounds.

Finally, by considering this formal approach, each control application (following different technologies) becomes a network of *homogeneous* components distributed on several containers of a controller. According to specifications, these components have to respect bounds on their response times.

Running example. *In all the continuation, we consider as an example a control application embedded in a vehicle. This application is composed of the following sub-applications to distribute on three containers :*

- *The temperature regulator developed while following the IEC 61499 technology.*
- *The speed regulator developed while following the PBO technology.*
- *The brake system developed while following the Rubus technology.*

To deploy the application in OS tasks of the execution support, we have to transform the corresponding heterogeneous components into formal homogeneous ones (figure 7). In the container 1 containing components controlling the break system, we distinguish two interactions : $act\_out\_left \mid act\_left$ *and* $act\_out\_right \mid act\_right$. *In this container, when the input actions (act_pres and act_speed) of* $Comp1$ *are activated, we have to execute the internal action* $Act1$ *deducing the brake to activate. Once the execution ends, we activate* act_out_left ***OR*** act_out_right *depending on the data pressure and speed.*

5 SUBTASKS SYSTEM

Once the heterogeneous components of the application are unified in a same model, the remaining problem is to deploy them in feasible OS tasks of the execution support. We propose to transform first of all these components into a subtasks system with precedence constraints. The purpose is to exploit previous results on the real-time scheduling. We define the following concepts characterizing such system.

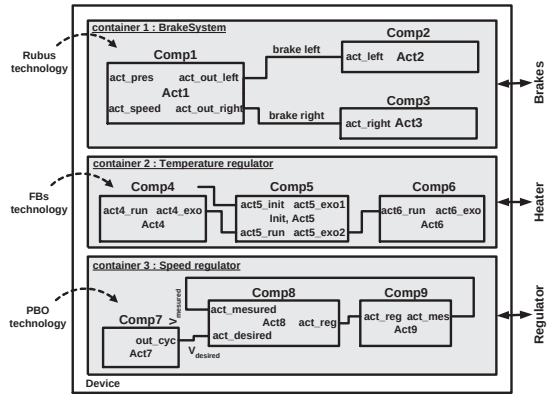

Figure 7: Distribution of the formal components on containers of the execution support.

5.1 Subtask

An application subtask, denoted by *sub*, corresponds to an execution of a component *c* when corresponding input actions are activated. In addition to these actions, the subtask *sub* implements the corresponding internal and output actions of the component. To generate the different subtasks of a component, we have quite simply to analyze the corresponding behavioral model. In this paper, we propose the following functions defining a subtask of a component.

- $cause\_of(sub)$: defines the input actions activating the execution of *sub*.
- $code\_of(sub)$: defines the internal actions to execute once the $cause\_of(sub)$ actions are activated.
- $effect\_of(sub)$: defines supersets of output actions. These sets are generated while applying an analysis of the interaction model. Each set represents a possible execution scenario of the component. At run-time, we have to execute only one of them depending on the execution of $code\_of(sub)$ actions.

Running example. *In the example, we define for the component Comp*1 *a subtask Sub*1*. We characterize this subtask as follows,*

- $cause\_of(sub) = \{act\_pres, act\_speed\}$
- $code\_of(sub) = \{Act1\}$
- $effect\_of(sub) = \{\{act\_out\_left\}, \{act\_out\_right\}\}$. *Once the execution of Act*1 *finishes, we have to execute act_out_left OR only act_out_right.*

Let Σ be the set of the application subtasks. We propose to characterize a component subtask *sub* as follows,

- $WCET(sub)$ (resp, $BCET(sub)$) : the Worst (resp, Best) Case Execution Times of the algorithms implementing the actions belonging to $cause\_of(sub)$, $code\_of(sub)$ and $effect\_of(sub)$.
- $pred(sub)$: a set of sub-tasks to execute in the application before *sub*. These subtasks belong to components that contain the output actions activating those of $cause\_of(sub)$.

$$pred(sub) = \{sub' \in \Sigma / \exists ia \in cause\_of(sub), \exists \psi \in effect\_of(sub'), \exists oa \in \psi, oa \in cause(ia)\}$$

- $succ(sub)$: a set of subtasks sets. Each subtasks set corresponds to a possible execution scenario (ie. only one subtasks set between all ones is executed at run-time). The subtasks of a set have to be executed once the execution of *sub* is finished.

$$succ(sub) = \{\phi \subset \Sigma / \exists \psi \in effect\_of(sub), \forall oa \in \psi, \exists sub' \in \phi, \exists ia \in cause\_of(sub'), oa \in cause(ia)\}$$

Let σ be a subset of Σ. This subset corresponds to a particular container of the application. We denote by first(σ) (resp last(σ)) the set of subtasks with no predecessors (resp successors) in σ. In this paper, we propose particularly to characterize each subtask *sub* of $first(\sigma)$ by a release time $r(sub)$ and a period $p(sub)$.

Running example. *In the followed example, the subtask sub*1 *is with no predecessors. In addition, once it is executed, we have to execute the subtasks sub*2 *OR sub*3 *depending on the pressure and the Speed.*

$$pred(sub1) = \emptyset; pred(sub2) = \{sub1\}$$
$$succ(sub1) = \{\{sub2\}, \{sub3\}\}$$

5.2 Subtasks Trace

By considering the precedence constraints between subtasks, we define a trace *tr* of σ the following sequence,

$$tr = sub_0, sub_1 ...., sub_{n-1}$$

such as,

- $\forall sub_i \in [1, n-1], sub_{i-1} \in pred(sub_i)$.
- $sub_0 \in first(\sigma)$ and $sub_{n-1} \in last(\sigma)$.

The trace *tr* implements a possible execution scenario of σ. Such execution has to respect a particular end to end response time bound (between the activation of sub_0 and the execution end of sub_{n-1}) according to specifications.

Running example. *In the same followed example, we distinguish two traces in the first container.*

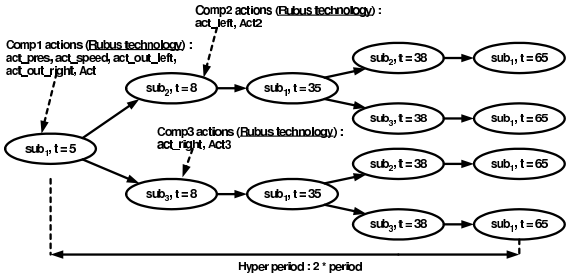

Figure 8: Temporal validation of the container1 containing Rubus components

$$trace_1 = sub_1, sub_2 \text{ and } trace_2 = sub_1, sub_3$$

6 DEPLOYMENT OF COMPONENTS

Once the application components are correctly transformed into a system of subtasks with precedence constraints, the problem is to deploy them in feasible OS tasks of a PLC. We propose to apply the approach particularly proposed in (Khalgui et al., 2006) for the deployment of systems of real-time subtasks. According to this approach, we apply a schedulability analysis on each container to check the internal blocks. This analysis is based on a generation of an accessibility graph that we propose. Once all the containers are feasible, we propose to transform them into independent OS tasks before checking their on-line preemptive feasibility (Khalgui et al., 2006).

Running example. *In the followed example, we suppose periodic readings of the pressure and the speed values (the release time $r = 5$ and the period $p = 30$). Moreover, we suppose that the worst and best case execution times of each action act $(act \in \Sigma)$ are bcet(act) = wcet(act) = 1. Therefore, $BCET(sub_1)$ = $WCET(sub_1)$ = 3. In the figure 8, we present the accessibility graph corresponding to the container 1. Once the container 1 is temporally validated, we construct a corresponding OS task that implements the different execution scenarios of the internal components (figure 9). The processed sequencing function is the main() function of this task where each subtask implements a set of application actions (Khalgui et al., 2006).*

7 CONCLUSION

This paper deals with the deployment of industrial control applications following different component-based technologies. The purpose is to reduce the realization time by exploiting the advantages and the different libraries of these technologies. To deploy the application on a multi-tasking PLC while satisfying the specifications constraints, we propose to fix a particular formal component-based approach in the order to cover all the known technologies. We propose also to enrich this approach to be compliant with these technologies. Therefore, the heterogeneous components of the application are considered as homogeneous formal ones. We propose to transform them into a subtasks system with precedence constraints to construct corresponding feasible OS tasks in the PLC.

In the future works, we plan to extend our research by considering re-configurable applications following different technologies. We have, first of all, to define the different reconfiguration scenarios of the heterogeneous components of the application. Then we have to study their deployment for each scenario case.

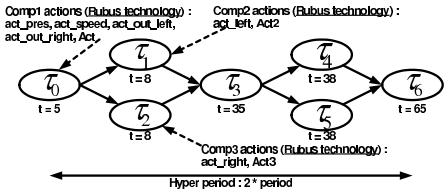

Figure 9: OS task implementing the first container containing Rubus components.

REFERENCES

Crnkovic, I. and Larsson, M. (2002). *Building reliable component-based software systems*. Artech House. London.

Khalgui, M., Rebeuf, X., and Simonot-Lion, F. (2006). Component-based deployment of industrial control systems : an hybrid scheduling approach. In *ETFA06, Czech*.

Lewis, R. (2002). *Modelling Control Systems using IEC61499*. The institution of Electrical Engineers.

Sifakis, J. (2005). *A Framework for Component-based Construction*. 3rd IEEE International Conference on Software Engineering and Formal Methods (SEFM05).