Proceedings of the 13th IFAC Symposium on
Information Control Problems in Manufacturing
Moscow, Russia, June 3-5, 2009

Refactoring of Execution Control Charts in Basic Function Blocks of the IEC 61499 Standard

Dubinin, V.*, Vyatkin, V.**

** Computer Science Department, Penza State University,
Penza, Russia (Tel.: +7-841-236-8227; e-mail: victor_n_dubinin@yahoo.com)
** Department of Electrical and Computer Engineering, University of Auckland,
Auckland, New Zealand (e-mail: v.vyatkin@auckland.ac.nz)}*

Abstract: This paper deals with refactoring of execution control charts of IEC 61499 basic function blocks as a means to improve the engineering support potential of the standard in development of industrial control applications. The main purpose of the refactoring is removal of arcs without event inputs and getting rid of potential deadlock states. The ECC refactoring is implemented as a set of graph transformation rules. A prototype has been implemented using the AGG software tool.

Keywords: refactoring, IEC 61499, graph transformation, function block, discrete control

1. INTRODUCTION

The international standard IEC 61499 defines a component-based architecture for the new generation of distributed component control systems [IEC 61499 (2005)]. The main artefact for creating applications as per the standard is a function block (FB). Many ideas of system engineering with function blocks can be borrowed from software engineering.

Model Driven Engineering – (MDE) is one of the state-of-the-art software engineering technology, and it operates with models and their transformations [Sendall & Kozaczynski (2003)]. The Object Management Group (OMG) [OMG Web-site] has proposed the Model Driven Architecture (MDA) for integration of various MDE tools. For definition of models and metamodels the OMG consortium has developed standards MOF and UML. In [Ledeczi et al. (2001)] the approach, called Model-Integrated Computing (MIC) for expanding MDA into the field of domain-specific modelling languages is proposed. MIC was applied in [Thramboulidis (2005)] in the area of mechatronic systems.

Graph transformations [Ehrig et al. (1999)] are a promising technique of implementing model transformations, as confirmed by its application in MDE, e.g. [Grunske et al. (2005)]. According to us, this approach is also appropriate for use in engineering of distributed component-based control systems with the new international standard IEC 61499. Main artefacts of the standard, such as composite FBs, applications and subapplications, can be represented in an abstract graph form. This also applies to basic FBs whose Execution Control Chart (ECC) can be naturally represented as a graph.

Model refactoring is an important direction in the field of model transformations. In a broad sense the refactoring changes program structure without changing its semantics. Refactoring is a technique supporting evolution of software systems, which can be applied to different abstraction levels of software models – from low-level code up to high level models. A good introduction to refactoring using graph transformation can be found in [Mens (2006)].

In this paper we solve the problem of ECC diagrams refactoring in basic FBs. The main purpose is to get rid from arcs having no event conditions and from (conditionally) dead states. Refactoring in this case is largely based on the notion of reachability of EC actions' sequences. The graph transformation rules for the ECC refactoring are presented. The prototype refactoring system is implemented in the graphs transformation tool AGG [Taenzer (2000)].

The paper is structured as follows. In Section 2 a formal model of ECC syntax is introduced. Section 3 discusses ECC execution rules. The concept of ECC refactoring is defined in Section 4. Section 5 presents the refactoring by means of graph transformations, and Section 6 discusses its implementation using the AGG software tool. The paper is concluded with a short summary and references.

2. MODEL OF EXECUTION CONTROL CHART

The Execution Control Chart (*ECC*) is a state machine determining sequence of operations in a basic FB [IEC 61499 (2005)]. For the purposes of refactoring we use a simplified model *of ECC,* different from [Dubinin & Vyatkin (2006)].

Let's define *ECC* as a tuple: $ECC = (S, R, E, C, A, f_E, f_C, f_A, f_P)$, where:

$S=\{s_1, s_2,\dots, s_n\}$ is a set of the vertices representing *EC-states*;

$R \subseteq S \times S$ is a set of the arcs representing *EC-transitions*;

$E= \{e_1,e_2,\dots,e_m\}$ is a set of event inputs;

$C=\{c_1,c_2,\dots,c_k\}$ is a set of guard conditions defined over input, internal and output variables of a basic FB;

$A= \{a_1,a_2,\dots,a_p\}$ is a set of *EC-actions'* sequences.

The set of arcs R is divided into class-rooms: R_E - event, R_C - conditional and R_T - unconditional arcs, $R = R_E \cup R_C \cup R_T$; $R_E \cap R_C \cap R_T=\varnothing$.

According to the standard, the syntax of EC-transition conditions is defined as: *Event input* | *Guard condition w/out event inputs* | *Event input & Guard.*

An event arc *(E-arc)* represents *EC-transition* with event input in its condition, a conditional arc *(C-arc)* an *EC-transition* without event input, having a guard condition with non-constant *true* value, and unconditional arc *(T-arc)* having no event inputs and constant *true* value. In the following we shall designate *E*-and *T*-arcs with a solid line, and *C-arc* with a dashed line. When necessary, in drawings we shall put a character *"t"* above *T-arcs*, and *"e"* above *E*-arcs.

f_E: $R_E \to E$ — the function assigning event inputs to *E*-arcs;

f_C: $R_E \cup R_C \to C$– assignment o f guard conditions *to E-*,*C*-arcs;

f_A: $S \to A$ – assignment of *EC-actions'* sequences to states;

f_P:$R \to D$ - the function, assigning priorities to arcs, where $D = \{d_1, d_2, \ldots\}$ is a countable, linearly-ordered set of priorities with an order relation $\prec$. For two arbitrary arcs $r_1, r_2 \in R$, *if* $f_P(r_1)=d_i$ and $f_P(r_2)=d_j$ and $d_i \prec d_j$ (in our case, $i<j$), then priority of the arc r_1 is considered to be higher than of the arc r_2. For convenience we shall use normalized priorities, defined as follows. Let R^s is the set of all arcs which are starting in the vertex *s*. The function of assigning normalized priorities of arcs for the vertex *s* is defined as follows: f_P^s:$R^s \to \{1,2,\ldots,|R^s|\}$, and the general function of prioritization (for the whole ECC) is defined as $f_P = \bigcup_{s \in S} f_P^s$.

In *ECC* of *IEC* 61499, priorities *of EC-transitions* are not defined explicitly, instead, the priority is based on the location of the transition in the textual representation of the function block (in XML format).

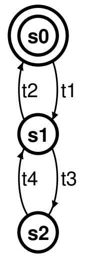
3. MODELS OF ECC EXECUTION

According to *IEC* 61499, an *ECC* is interpreted following the state-machine presented in Table 1. The ECC interpreter is activated by an input event and continues evaluation of ECC until no EC-transition can clear. This process may include several EC-transitions and is called a single run of FB.

As it was noted in [Sünder et al. (2006)], [Vyatkin & Dubinin (2007)], the definition of ECC interpretation in the standard is incomplete and thus, ambiguous. It, for example, admits two different approaches to evaluation *of EC-transitions* without events.

According to the first approach, an *EC-transition* without events can be cleared only if it is not first in the run, but follows some other *EC transition* with event. The second approach does not link *EC-transition* to any concrete event. In this case enableness of the *EC-transition* is determined only by the value of its guard condition. We shall name an eventless guard condition in the first case *passive*, and the second case - *active*. Both approaches were studied in the literature. The first approach is presented in [Sünder et al. (2006)], the second is presented in the work [Vyatkin & Dubinin (2007)], introducing sequential model of FB execution. In the following we shall consider only the first model of *ECC* realization in which there is a direct necessity in refactoring of *ECC*.

Table 1. ECC operation state machine (Table 1 from [IEC 61499 (2005)], 5.2.2)



State		Operations
s0		---
s1		Evaluate transitions
s2		Perform actions
Transition	**Condition**	**Operations**
t1	Invoke ECC	Sample inputs
t2	No transition clears	
t3	A transition clears	
t4	Actions completed	

Definition 1. Potentially-deadlock (PD) is an ECC state if all arcs going out it are conditional.

Assertion. If *ECC* interpreter made transition *t2* (Table 1) while *ECC* was in a PD-state then this state becomes a deadlock. No event signal can change this state.

Definition 2. Two *ECCs* are called *functionally equivalent* (within the limits of a certain model of ECC execution), if at any sequences of input events and corresponding values of input variables, both *ECCs* follow the same sequences *of EC-actions*.

4. REFACTORING OF ECC

Refactoring of *ECC* is used for getting rid of: 1) C-*arcs*; 2) PD-states. According to these goals we will distinguish two types of refactoring (type 1 and 2). Results of refactoring 2 are based on the results of refactoring 1. Refactoring 2 has direct practical significance. At the same time, refactoring 1 can help the developer to have a different point of view on developed *ECC* that in some cases (on the basis of the visual analysis) can help to rethink and redesign it.

Let's name *CT-network* of *ECC* a subgraph containing arcs only from $R_C \cup R_T$; but not from R_E. Generally the given graph may be not connected. Accordingly, as *T-network* of *ECC* we shall name a subgraph containing all arcs from R_T.

Let's introduce $ES = \{(s,s') \in R_E \mid \exists (s',s'') \in R_C \cup R_T\}$ – the set of the *E-arcs* forming a path of length 2 with one of *C-* or *T-arcs* of a *CT-network*. Let us name these arcs as *sources*. It is assumed, that the initial *CT*-network is acyclic. Presence of cycles tells about incorrectness of the *ECC*.

The general idea of removing *C*-arcs from *ECC* is based on the concept of reachability of sequences of *EC-actions* at the *ECC* interpretation. Let $(s_0, s_1) \in ES$ be an *E*-arc followed by the path $s_1, s_2, \ldots, s_k$ in the *CT*-network. For each *EC-state* s_i ($i=\overline{1,k}$) there is a sequence of associated *EC-actions* a_i. An example is given in Fig. 1, where the path w.l.o.g. consists of *C*-arcs only.

This path can be substituted by one *E-arc* (s_0, s_k) whose guard condition is composed from the guard conditions of the arcs (c_i, $i=\overline{1,k}$), constituting the path, and from the so-called