

В.Н.Дубинин, В.В.Вяткин

ФОРМАЛИЗОВАННОЕ ОПИСАНИЕ И МОДЕЛИРОВАНИЕ СИСТЕМ ФУНКЦИОНАЛЬНЫХ БЛОКОВ IEC 61499

В работе дается формализованное описание системы типов программных объектов для определения приложений IEC 61499, рассматривается переход от системы типов к системе экземпляров, определяется модель функциональных блоков, приводятся правила ее функционирования. Для формальных определений используется теоретико-множественный и структурно-лингвистический подход.

Введение

Принятый в этом году новый международный стандарт *IEC 61499* [1] определяет путь для создания систем управления промышленными процессами нового поколения. Это интеллектуальные распределенные компонентно-базированные системы на основе функциональных блоков (ФБ). Особенности ФБ являются: машина состояний потоков событий (называемая диаграммой *ECC- Execution Control Chart*), входные и выходные событийные переменные с собственным управлением, входные и выходные переменные для представления потоков данных, а также ориентация функциональных блоков на реализацию (ФБ - выполняемая спецификация). Установка (*sampling*) значений входных и выходных переменных производится с помощью событийных входов и выходов, соответственно. Для этого используются специальные *WITH*-связи. С состояниями *ECC* могут быть связаны выходные события и алгоритмы, определенные на одном из языков программируемых контроллеров стандарта *IEC 61131-3* [2]. В соответствии со стандартом *IEC 61499* управляющее приложение представляется в виде сети связанных между собой ФБ, которые могут выполняться на различных ресурсах и устройствах системы.

В стандарте *IEC 61499* приводятся неформальные или полужформальные определения ФБ, затрагивающее только концептуальные аспекты. Этого вполне достаточно для пользователей и потребителей систем на основе ФБ, включая инженеров по управлению, системных интеграторов, менеджеров и т.д. Однако для построения систем автоматизированного проектирования распределенных компонентно-базированных систем управления промышленными процессами (РКБСУПП) этого явно недостаточно и требуются более формальные определения и модели.

В настоящее время существует небольшое число работ, в которых затрагиваются вопросы формального описания и моделирования ФБ стандарта *IEC 61499*. Работа [3] посвящена моделированию ФБ с использованием *NCES*-формализма, являющегося расширением сетей Петри. Авторы работы [4] предложили модель,

основанную на синхронном языке потоков данных *Signal* и связанном с ним средстве верификации стандарта *IEC 61499*. В работе [5] для моделирования поведения базисных ФБ используются временные автоматы. Существует несколько работ, в числе которых работы [6,7], в которых для моделирования ФБ предлагается использовать язык *UML*. Однако, несмотря на все преимущества языка *UML*, следует учитывать, что работы по определению его формальной семантики еще не закончены [8].

В данной работе предлагается формальная модель ФБ, которую концептуально можно отнести к моделям переходов состояний. В отличие от известных моделей ФБ предлагаемая модель ориентирована на внешнюю (представительную) логику ФБ, выражаемую диаграммами *ECC* (учитывая условия перехода и алгоритмы) и связями между ФБ, а не на внутреннюю, затрагивающую детали реализации.

В первом разделе данной работы приводится формализованное описание системы типов программных объектов для определения приложений *IEC 61499*, в том числе описываются типы базисного и составного ФБ, а также субприложения и приложения. Здесь же описывается процесс перехода от системы типов к системе экземпляров. Второй раздел посвящен определению модели ФБ. В нем отмечаются особенности модели, приводятся семантические модели интерфейсов ФБ, рассматриваются клапаны данных (КД) и вопросы приведения модели к каноническому виду. Подробно рассматривается динамика функционирования предложенной модели, при этом вводятся определения состояний системы и правила перехода между состояниями.

1. Система типов программных объектов для определения приложений

Определим систему (или базу данных) типов программных объектов *FBT*, из которых строятся конкретные управляющие приложения в соответствии со стандартом *IEC 61499*, следующим образом:

$$FBT = (FBType, FBKind, FBName),$$

где *FBType* – множество определений типов объектов;

FBKind: FBType → {*bfb, cfb, sifb, subappl, appl*} – функция, ставящая в соответствие каждому определению типа объекта сорт типа. Существуют следующие сорта типов: *bfb* – базисный ФБ; *cfb* – составной ФБ; *sifb* – сервисный интерфейсный ФБ (СИФБ); *subappl* – субприложение; *appl* – приложение. Кроме этих стандартных сортов в дальнейшем вводится специальный сорт *dv* – клапан данных, используемый при моделировании. В соответствии с функцией *FBKind* все множество определений типов можно разбить на непересекающиеся классы:

$$FBType = BFBType \cup CFBType \cup SIFBType \cup SubApplType \cup ApplType$$

Смысл классов понятен из приведенных выше названий. Каждый сорт типов объектов имеет свой шаблон (схему) для определения типов. При использовании вложенных кортежей считается, что сделанные в них определения наследуются во внешних и соседних кортежах.

FBName: FBType → *Name* – функция, ставящая в соответствие каждому типу имя из домена имен *Name*.

Определение типа базисного функционального блока

Тип базисного ФБ определяется следующим кортежем

$$(Interface, ECC, V),$$

где *Interface* – интерфейс объекта;

$$ECC – \text{диаграмма } ECC;$$

$V = \{v_1, v_2, \dots, v_p\}$ – множество внутренних переменных.

Интерфейс объекта определяется следующим кортежем:

$$Interface = (EI^0, EO^0, VI^0, VO^0, IW, OW),$$

где $EI^0 = \{ei_1^0, ei_2^0, \dots, ei_{k0}^0\}$ – множество событийных входов;

$EO^0 = \{eo_1^0, eo_2^0, \dots, eo_{l0}^0\}$ – множество событийных выходов;

$VI^0 = \{vi_1^0, vi_2^0, \dots, vi_{m0}^0\}$ – множество информационных входов;

$VO^0 = \{vo_1^0, vo_2^0, \dots, vo_{n0}^0\}$ – множество

информационных выходов;

$IW \subseteq EI^0 \times VI^0$ – множество *WITH*-связей для входов;

$OW \subseteq EO^0 \times VO^0$ – множество *WITH*-связей для выходов.

Для корректности задания интерфейса должны выполняться условия:

$$VI^0 \setminus Pr_2 IW = \emptyset \text{ и } VO^0 \setminus Pr_2 OW = \emptyset$$

Иными словами, каждый информационный вход и выход должен быть инцидентен какой-либо *WITH*-связи. Пример интерфейса ФБ приведен на рис. 1.

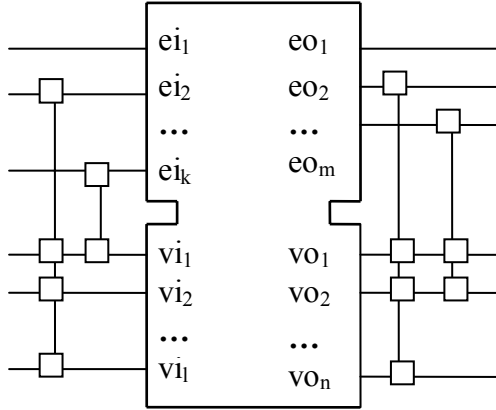


Рис.1. Синтаксическая модель интерфейса функционального блока

выполнения алгоритмов, содержащихся в ФБ. Данная автоматная модель в стандарте названа диаграммой управления выполнением функционального блока (*ECC* – Execution Control Chart).

При определении диаграммы *ECC* будут использоваться следующие обозначения. Множество всех функций, отображающих множество A в множество B будем обозначать как $[A \rightarrow B]$. В некоторых случаях индекс элементов множеств может опускаться, если это не вызывает неопределенности. Под $Dom(x)$ будем понимать множество допустимых значений переменной x .

Диаграмма *ECC* может быть определена следующим образом:

$$ECC = (State, Tran, Cond, OutEv, Alg, s0),$$

где $State = \{s_0, s_1, s_2, \dots, s_r\}$ – множество состояний диаграммы *ECC*;

$Tran \subseteq State \times State$ – множество переходов состояний;

$$Cond : Tran \rightarrow \left[\prod_{ei \in EI^0} (\{ei\} \times \{0,1\}) \times \prod_{vi \in VI^0} (\{vi\} \times Dom(vi)) \times \right.$$

$$\left. \prod_{vo \in VO^0} (\{vo\} \times Dom(vo)) \times \prod_{v \in V} (\{v\} \times Dom(v)) \rightarrow \{true, false\} \right]$$

- функция, назначающая переходам условия переходов в виде логических выражений. Как видно из формулы, логическое выражение вычисляется на множестве входных, выходных и внутренних переменных, а также входных событийных переменных;

$$Alg : State \rightarrow \left[\prod_{vi \in VI^0} (\{vi\} \times Dom(vi)) \times \prod_{vo \in VO^0} (\{vo\} \times Dom(vo)) \times \prod_{v \in V} (\{v\} \times Dom(v)) \right. \\ \left. \rightarrow \prod_{vo \in VO^0} (\{vo\} \times Dom(vo)) \times \prod_{v \in V} (\{v\} \times Dom(v)) \right]$$

- функция, назначающая состояниям алгоритмы. Как видно из определения, алгоритмы могут изменять только внутренние и выходные переменные;

$OutEv: State \rightarrow P(EO^0)$ – функция, назначающая переходам совокупность выходных событий (точнее, выходных событийных линий). Здесь символ P используется для обозначения булеана множества;

$s0 \in State$ – начальное состояние.

Определение типов составного функционального блока, субприложения и приложения

Типы составного ФБ и субприложения определяются следующим кортежем

$(Interface, FBI, FBType, EventConn, DataConn)$,

где $Interface$ – интерфейс (определен выше). Особенностью интерфейса субприложения является то, что в нем отсутствуют $WITH$ -связи, то есть $IW=OW=\emptyset$;

$FBI = \{fbi_1, fbi_2, \dots, fbi_n\}$ – множество ссылочных экземпляров ФБ, входящих в тип.

Графическое представление ссылочного экземпляра сходно с графическим интерфейсом типа субприложения. Каждый ссылочный экземпляр $fbi_j \in FBI$ определяется четверкой следующих множеств:

$EI^j = \{ei_1^j, ei_2^j, \dots, ei_{k_j}^j\}$ – множество событийных входов;

$EO^j = \{eo_1^j, eo_2^j, \dots, eo_{l_j}^j\}$ – множество событийных выходов;

$VI^j = \{vi_1^j, vi_2^j, \dots, vi_{m_j}^j\}$ – множество информационных входов;

$VO^j = \{vo_1^j, vo_2^j, \dots, vo_{n_j}^j\}$ – множество информационных выходов.

$FBType: FBI \rightarrow FBType$ – функция, ставящая в соответствие каждому ссылочному экземпляру ФБ тип. При этом интерфейс ссылочного экземпляра должен соответствовать интерфейсу сопоставленного ему типа. Следует отметить, что на практике (при нисходящем проектировании) часто встречается ситуация, когда ссылочному экземпляру не сопоставлен ни один тип. Более точно областью значений функции $FBType$ для (типа) составного ФБ является множество $BFBType \cup CFBType \cup SIFBType$. Для (типа) субприложения это множество дополняется множеством $SubApplType$. Это связано с тем, что субприложение может быть размещено на нескольких ресурсах, а составной ФБ – только на одном.

$EventConn \subseteq (\bigcup_{j \in I, n} EO^j \cup EI^0) \times (\bigcup_{j \in I, n} EI^j \cup EO^0)$ – множество событийных связей;

$DataConn \subseteq (VI^0 \times \bigcup_{j \in I, n} VI^j) \cup (\bigcup_{j \in I, n} VO^j \times (\bigcup_{j \in I, n} VI^j \cup VO^0))$ – множество информационных связей.

Причем для информационных связей должно выполняться условие

$$\forall (p, t), (q, u) \in EventConn [(t = u) \rightarrow (p = q)]$$

Иными словами, к одному информационному входу нельзя подключить более одного информационного выхода. При использовании событийных связей подобных топологических ограничений на структуру не накладывается, поскольку подразумевается неявное использование блоков E_SPLIT и E_MERGE для расщепления и слияния событий, соответственно.

Переход от системы типов к системе экземпляров

Переход от системы типов к системе экземпляров сводится к замене ссылочных экземпляров объектов на соответствующие им реальные экземпляры, представляющие содержимое объекта. Реальные экземпляры получаются путем клонирования описания типа, соответствующего ссылочному объекту. Замена производится, начиная с некоторого начального типа, в качестве которого обычно выступает тип приложения.

Синтаксически экземпляр ФБ является копией соответствующего типа. Поэтому в дальнейшем при описании экземпляров ФБ будем пользоваться введенными обозначениями из описания соответствующих типов.

Система экземпляров полностью определяется деревом иерархии экземпляров, которое можно определить следующим кортежем.

$(F, Aggr, FBType_A, FBId_A)$,

где F – множество (реальных) экземпляров ФБ и субприложений;

$Aggr \subseteq F \times F$ – отношение агрегации;

$FBType_A: F \rightarrow FBName$ – функция разметки узлов дерева именами типов ФБ;

$FBId_A: F \rightarrow Id$ – функция разметки узлов дерева уникальными идентификаторами из домена Id .

Ниже приводится алгоритм построения дерева иерархии экземпляров.

Рекурсивная процедура $expand(f)$ развертывает все входящие в реальный экземпляр f ссылочные экземпляры (любого уровня).

procedure $expand(f)$

if $KindOf(f) \in \{cfb, subappl, appl\}$ **then**

do forall $fbi \in FBI(FBType_A(f))$

$newF = InstanceOf(FBType(fbi))$

Заменить fbi на $newF$

$F = F \cup \{newF\}$

$Aggr = Aggr \cup \{(f, newF)\}$

$FBType_A = FBType_A \cup \{(f, FBName(FBType(fbi)))\}$

$FBId_A = FBId_A \cup \{(f, NewId())\}$

$expand(newF)$

end forall

end if

end procedure

В приведенной процедуре используются следующие вспомогательные функции. Функция $InstanceOf$ формирует экземпляр заданного типа. Функция $KindOf$ служит для определения сорта типа заданного экземпляра. Функция FBI определяет множество ссылочных экземпляров для заданного типа. Функция $NewId$ создает новый уникальный идентификатор для созданного экземпляра.

Замена ссылочного экземпляра реальным производится в три этапа: 1) добавление реального экземпляра; 2) встраивание реального экземпляра; 3) удаление ссылочного экземпляра. Встраивание реального экземпляра производится путем перекоммутации событийных и информационных линий с входов-выходов ссылочного экземпляра на соответствующие входы-выходы реального экземпляра. Между интерфейсами реального и ссылочного экземпляров должно существовать взаимно однозначное соответствие.

Построение дерева экземпляров начинается с некоторого начального типа fbt_0 :

$f_0 = InstanceOf(fbt_0)$; $F = \{f_0\}$; $Aggr = \emptyset$; $FBType_A = \{(f_0, FBName(fbt_0))\}$; $FBId = \{(f_0, NewId())\}$; $expand(f_0)$

2. Модель функциональных блоков

Ниже приводится формальная модель ФБ, определенная в рамках моделей переходов состояний. К данному виду моделей относятся конечные автоматы, формальные грамматики, сети Петри и другие модели. Используемый здесь для краткости термин “функциональные блоки” используется в широком смысле. Под ФБ понимаются все программные объекты, из которых строится приложение.

В отличие от известных моделей ФБ предлагаемая модель ориентирована на внешнюю (представительную) логику ФБ, выражаемую диаграммами ЕСС (учитывая условия перехода и алгоритмы) и связями между ФБ, а не на внутреннюю, затрагивающую детали реализации. Предлагаемая модель проста и понятна. Она является упрощением реальной модели ФБ, представленной в стандарте *IEC 61499* [1]. За основу модели взяты синтаксические конструкции ФБ стандарта *IEC 61499*, расширенные семантической информацией, полученной непосредственно из самого стандарта, или предложенной авторами на основе анализа стандарта.

При задании модели ФБ определяются элементы, состояния элементов и правила перехода из одного состояния в другое. Кроме того, определяются правила перехода от программных объектов стандарта *IEC 61499* к объектам, определенным в модели.

Особенности модели представлены ниже.

- 1) не рассматриваются временные аспекты, например, считается, что алгоритмы выполняются мгновенно;
- 2) в связи с принятием п.1 исчезают потери при приеме входных событий, а также конфликты из-за ресурсов, входящих в среду выполнения ФБ;
- 3) основным и единственным действием, переводящим модель из одного состояния в другое, является срабатывание перехода диаграммы *ЕСС* в одном из базисных ФБ;
- 4) используется групповое правило срабатывания переходов внутри одного базисного ФБ, что позволяет избежать рассмотрения машины управления выполнением *ЕСС*;
- 5) в модели используются буфера данных, хотя в стандарте они явно не фигурируют;
- 6) в модели функционирования рассматриваются только такие программные объекты, как приложение, базисный ФБ и КД;
- 7) не рассматриваются некоторые детали, относящиеся к технологической цепочке обработки событий и запуска/завершения *ЕСС*, в частности, рассматривается только однокаскадная схема приема входных событий;
- 8) все алгоритмы, связанные с состоянием, выполняются в виде одного обобщенного действия, причем данное действие должно учитывать порядок вызова алгоритмов. Это является определенной абстракцией, позволяющей отказаться от рассмотрения функции диспетчирования (*Scheduling Function*);
- 9) синхронное выполнение всех необходимых действий при срабатывании перехода *ЕСС* позволяет существенно сократить число несущественных промежуточных состояний системы, что важно для целого ряда методов верификации.

Приведенные особенности и абстракции требуют учета сферы и целей применения модели.

Семантические модели интерфейсов

При моделировании систем ФБ необходимо учитывать семантику синтаксических конструкций. В модели используется следующая семантическая интерпретация интерфейсных элементов.

- 1) каждому событийному входу базисного ФБ соответствует входная событийная переменная;
- 2) каждому информационному входу базисного или составного ФБ соответствует входная переменная;
- 3) каждому информационному выходу базисного ФБ соответствует выходная переменная и сопряженный с ним однотипный буфер данных;
- 4) каждому информационному выходу составного ФБ соответствует буфер данных;

5) информационным входам и выходам субприложения переменные не сопоставляются;

6) каждой константе на входе ФБ соответствует буфер данных.

Следует отметить, что в стандарте *IEC 61499* буферы данных явно не фигурируют. В нашей трактовке буфер данных (единичной емкости) служит для хранения данных, выдаваемых наружу ФБ с помощью соответствующего “стробирующего” сигнала (с использованием *WITH*-связи). Входные, выходные переменные, а также буфера данных могут принимать значения из соответствующих им доменов.

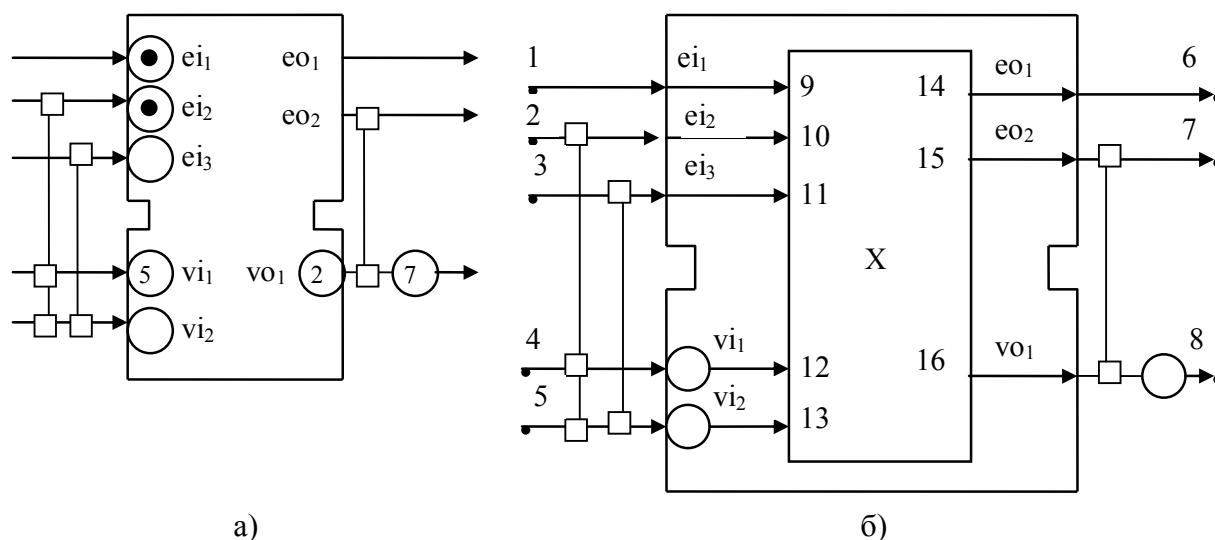


Рис. 2. Семантические модели интерфейсов функциональных блоков: а) базисного ФБ; б) составного ФБ

Для представления семантических моделей интерфейсов предлагается следующая графическая нотация. Все виды переменных и буфера данных представляются кружками, рисуемыми около соответствующих входов и выходов. Внутри кружка, соответствующего входной событийной переменной, может стоять точка, которая соответствует взведенному состоянию данной переменной (данная нотация заимствована из сетей Петри). Внутри кружков, соответствующих входным и выходным переменным, а также буферам данных, ставится значение данных объектов. На рис. 2 в качестве примера приведены семантические модели интерфейсов базисного и составного ФБ. На рис. 2,б под *X* подразумевается некоторая произвольная схема.

Приведение модели к каноническому виду

Для упрощения моделирования предлагается использование одноуровневого представления системы ФБ, эквивалентное исходной системе, включающей в общем случае составные ФБ [9]. Одноуровневое представление системы ФБ является каноническим представлением, на основе которого определяется модель переходов состояний. Оно включает только неделимые (терминальные) программные объекты и связи между ними. Для перехода к одноуровневому представлению вводится понятие клапана данных (КД). Функция КД – копирование значений из входных буферов данных в выходные. КД не входит в рамки стандарта *IEC 61499*, но может использоваться при моделировании.

Процесс перехода к одноуровневому представлению может быть адекватно описан с использованием аппарата порождающих графовых грамматик (ГГ) [10], определяемых как (Σ, A, R, Z) , где Σ - множество меток узлов (алфавит ГГ); $A \subset \Sigma$ - множество терминальных меток узлов (множество имен ссылочных экземпляров типа “базисный ФБ”, “сервисный интерфейсный ФБ” или “клапан данных”); $\Sigma \setminus A$ - множество нетерминальных меток узлов (множество имен ссылочных экземпляров

типа “составной ФБ” или “субприложение”); R - конечное множество продукций, представляемых в форме (d, D) , где $d \in \Sigma \Delta$, D - правая часть продукции, представляющая определения типов объектов из множества $FVType$, дополненное необходимыми клапанами данных (см. рис.2,б и 3); Z - начальный нетерминал (начальный тип). В терминах ГГ переход от многоуровневого представления к одноуровневому представляется как вывод графа в порождающей ГГ. При корректном задании ГГ язык, порождаемый ГГ, будет состоять из единственного графа с терминальными метками, представляющим одноуровневый эквивалент системы ФБ.

Существует два подхода к построению КД. Первый подход основан на использовании элементарных КД. В качестве синтаксической модели КД в этом случае может использоваться синтаксическая модель субприложения с учетом следующих структурных ограничений: 1) КД имеет по одному событийному входу и выходу; 2) число входных информационных линий равно числу выходных, причем это число должно быть больше или равно 1.

В случае использования элементарных КД буфера данных являются самостоятельными синтаксическими элементами, поскольку в общем случае их нельзя “прикрепить” ни к одному КД. На рисунках они отмечаются двойными окружностями. При преобразовании составного ФБ число КД определяется числом событийных линий данного составного ФБ, причем эти линии должны быть связаны WITH-связью хоть с одной информационной линией.

Второй подход основан на использовании сложных КД. В качестве синтаксической модели КД в этом случае может использоваться синтаксическая модель ФБ. При переходе к одноуровневому представлению КД полностью наследует входную часть интерфейса составного ФБ, включая WITH-связи. Выходная часть интерфейса КД в этом случае должна быть идентичной входной части.

В случае использования сложных КД буфера данных ассоциируются с информационными выходами КД также, как и в случае ФБ. В сложном КД WITH-связи используются для определения тех информационных входов, которые должны быть “скопированы” в соответствующие буфера данных, связанных с соответствующими информационными выходами.

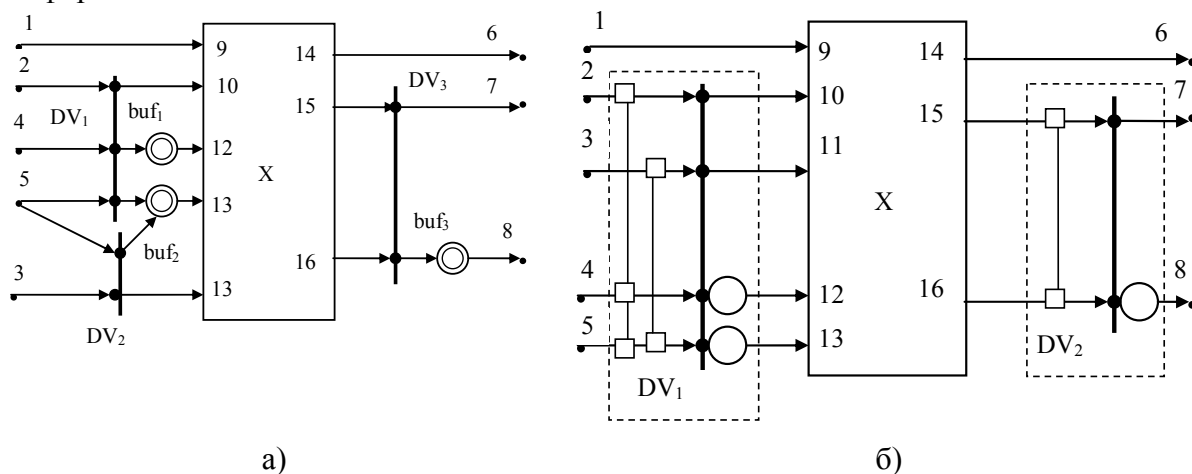


Рис. 3. Модель раскрытого составного функционального блока с использованием клапанов данных: а) первого типа; б) второго типа

Следует отметить, что с использованием отмеченных выше синтаксических моделей КД можно построить синтаксически правильное (в соответствии со стандартом) приложение. Однако при этом следует учитывать собственную семантику выполнения КД, отличную от программных единиц стандарта IEC 61499.

Предложенные выше синтаксические модели КД можно упростить, поскольку входы и выходы в КД являются сопряженными. При этом графическим изображением КД будет являться не прямоугольник, а отрезок прямой. На рис. 3 приведены

канонические представления составного ФБ, изображенного на рис. 2,б, с использованием КД обоих видов.

Модель функционирования приложений

Ниже приводится модель функционирования приложения, представляющего функционально законченную систему ФБ. Для упрощения при определении модели будем пользоваться следующими правилами использования индексов элементов множеств. Пусть задано множество A произвольно индексированных (двумя индексами) элементов, например, $A = \{a_k^s, a_n^p, \dots\}$. Для удобства будем допускать использование элемента множества как с указанием одного или двух индексов, или вообще без индексов. Например, элемент a множества A без индекса может представлять любой элемент данного множества с любым верхним и нижним индексом. Элемент множества A с одним индексом (например, a^i) может представлять любой элемент данного множества, у которого индекс совпадает с указанным и т.п..

Состояние приложения

Пусть приложение A состоит из программных объектов F_1, F_2, \dots, F_n , причем $FBTKind(TypeFBI(F_i)) \in \{bfb, dv\}$. Из этого следует, что приложение представляет одноуровневую сеть из базисных ФБ и КД. Состояние приложения в данном случае определяется как $SA = (SF^1, SF^2, \dots, SF^n)$, где SF^i – состояние i -го объекта. Если $FBTKind(TypeFBI(F_i)) = bfb$, то состояние SF^i определяется кортежем $SBFB$, а если $FBTKind(TypeFBI(F_i)) = dv$, то кортежем SDV .

Обозначим через EI^A, EO^A, VI^A, VO^A – соответственно множества входных и выходных событийных и информационных линий всех объектов, входящих в приложение A .

Состояние базисного функционального блока

Состояние базисного ФБ определяется кортежем вида (некоторые индексы в определении для краткости опускаются)

$$SBFB = (cs, ZEI, ZVI, ZVO, ZV, ZBUF),$$

где $cs \in State$ – текущее состояние диаграммы ECC ;

$ZEI: EI \rightarrow \{0, 1\}$ – функция значений входных событийных переменных. Если $EI(e_{ij}) = 1$, то входная событийная переменная e_{ij} установлена, иначе – сброшена;

$ZVI: VI \rightarrow Dom(VI)$ – функция значений входных переменных. Запись $ZVI(v_{ij}) = 5$ означает, что значение входной переменной v_{ij} равно 5;

$ZVO: VO \rightarrow Dom(VO)$ – функция значений выходных переменных;

$ZV: V \rightarrow Dom(V)$ – функция значений внутренних переменных;

$ZBUF: VO \rightarrow Dom(VO)$ – функция значений буферов данных.

Здесь для упрощения используется обозначение $Dom(X)$, где X – множество переменных - множество допустимых значений переменных из множества X .

Начальное состояние базисного ФБ определяется как

$$SBFB_0 = (s_0, ZEI_0, ZVI_0, ZVO_0, ZV_0, ZBUF_0),$$

где s_0 – начальное состояние диаграммы ECC ; $ZEI_0: EI \rightarrow \{0\}$. Функции $ZVI_0, ZVO_0, ZV_0, ZBUF_0$ определяются реализацией.

Состояние клапана данных

Состояние i -го КД (второго вида) определяется кортежем $SDV = (ZBUF)$, где функция $ZBUF$ определена выше.

Правило разрешенности перехода

Переход $(s_i, s_j) \in Tran$ диаграммы ECC базисного ФБ разрешен, если

$$(s_i = cs) \& \{Cond(s_i, s_j)\}(\overline{ZEI}, \overline{ZVI}, \overline{ZVO}, \overline{ZV}) = true,$$

где $\overline{ZEI}, \overline{ZVI}, \overline{ZVO}, \overline{ZV}$ - это, соответственно, функции ZEI, ZVI, ZVO и ZV , представленные набором пар вида (*переменная, значение*), например, $(vi_1, 5), (vi_2, 3), (vi_3, 7)$. Иными словами, переход разрешен, если его состояние-источник является текущим и условие перехода истинно. В фигурных скобках определена вычисляемая функция.

Правила срабатывания перехода

Определим несколько вспомогательных множеств. На уровне базисного ФБ введем множество $XEI(s_i, s_j) \subseteq EI$, представляющее входные событийные переменные, влияющие на разрешенность перехода (s_i, s_j) диаграммы ECC .

На уровне приложения определим следующие множества:

1) множество событийных входов базисных ФБ, являющихся приемниками событийных выходов из множества $S \subseteq EO^A$

$$EVPOST_{BFB}(S) = \bigcup_{eo \in S} evpost_{BFB}(eo),$$

где $evpost_{BFB}(eo) = \{ei^m \mid (eo, ei^m) \in EventConn \& FBType(F_m) = bfb\}$ - множество событийных входов базисных ФБ, являющихся приемниками событийного выхода $eo \in EO^A$;

2) множество $EVPOST_{DV}(S)$ событийных входов клапанов данных, являющихся приемниками событийных выходов из множества $S \subseteq EO^A$, определяется похожим образом;

3) множество информационных выходов, связанных $WITH$ -связями с событийными выходами из множества $E \subseteq EO^A$

$$EOWITH(E) = \bigcup_{e \in E} eowith(e),$$

где $eowith(eo^j) = \{vo^j \mid (eo^j, vo^j) \in OW^j\}$ - множество информационных выходов, связанных $WITH$ -связями с событийным выходом $eo^j \in EO^j$

4) Множество $EIWITH(E)$ информационных входов, связанных $WITH$ -связями с событийными входами из множества $E \subseteq EI^A$, определяется сходным образом.

5) множество источников информационного входа $vi \in VI^A$

$$dpre(vi) = \{vo \mid (vo, vi) \in DataConn\}$$

Новые функции состояний, полученные в результате срабатывания перехода, будем отмечать апострофом на месте верхнего индекса, например, ZEI' . Для обозначения обобщенных функций состояний для целого приложения A используется нижний индекс A , например, $ZBUF_A$. Обобщенные функции получают объединением соответственно областей определения, областей допустимых значений и графиков функций, определенных для базисных ФБ. Под текущим базисным ФБ будем понимать базисный ФБ, в котором сработал переход. Для обозначения функций текущего ФБ используется индекс s или функция используется без индекса.

Сработать может только разрешенный переход. При срабатывании перехода (s_i, s_j) выполняется неделимая последовательность действий:

- 1) $cs' = s_j$, то есть текущим состоянием становится состояние-приемник перехода
- 2) $\forall ei \in XEI(s_i, s_j) [ZEI(ei) = 0]$

Иными словами, сбрасываются в 0 те входные событийные переменные, которые фигурируют в условии перехода.

$$3) (\overline{ZV'}, \overline{ZVO'}) = \{Alg(s_j)\}(\overline{ZVI}, \overline{ZVO}, \overline{ZV})$$

Иными словами, алгоритм, присоединенный к состоянию-приемнику s_j , на основе текущих значений входных, выходных и внутренних переменных вырабатывает новые значения выходных и внутренних переменных.

$$4) \forall ei \in EVPOST_{BFB}(OutEv(s_j))[ZEI'_A(ei) = 1]$$

Иными словами, устанавливаются в 1 те входные событийные переменные окружающих базисных ФБ, которые связаны с событийными выходами текущего базисного ФБ, через которые выдаются выходные сигналы в случае, когда *ECC* текущего базисного ФБ переходит в состояние s_j .

$$5) \forall vo \in EOWITH(OutEv(s_j))[ZBUF'(vo) = ZVO(vo)]$$

Иными словами, значения выходных переменных текущего базисного ФБ, которые связаны *WITH*-связями с выходными событийными линиями, через которые выдавались сигналы, переписываются в сопряженные с ними буфера данных.

6)

$$\forall vi \in EIWITH(EVPOST_{BFB}(OutEv(s_j)))[ZVI'_A(vi) = ZBUF_A(dpre(vi))]$$

Иными словами, во входные переменные базисных ФБ, которые связаны *WITH*-связями с событийными входами, на которые подавались сигналы (в данный момент времени) с событийных выходов текущего базисного ФБ, диаграмма *ECC* которого перешла в состояние s_j , переписываются значения буферов данных, которые являются источниками этих входных переменных.

7)

$$\forall vi \in EIWITH(EVPOST_{DV}(OutEv(s_j)))[ZBUF'_A(vi) = ZBUF_A(dpre(vi))]$$

Иными словами, в буфера КД переписываются значения тех буферов данных, которые являются их предшественниками, причем буфера КД должны быть связаны *WITH*-связями с событийными входами, на которые подавались сигналы (в данный момент времени) с событийных выходов текущего базисного ФБ, диаграмма *ECC* которого перешла в состояние s_j . Следует, однако, заметить, что данное правило действительно только для случая, когда отсутствуют каскады КД. При наличии каскадов КД действие сложнее и основано на построении графов потоков событий и данных [9].

Функционирование приложения

Проиндексируем переходы *ECC* в каждом базисном ФБ, входящем в приложение A , с использованием независимой нумерации. Через $t_k^j \in T$ будем обозначать k -й переход в j -м базисном ФБ, где T - множество всех переходов состояний *ECC* всех базисных ФБ приложения A .

Обозначим $ENABLED_A(SA_i)$ множество разрешенных переходов приложения A в i -м состоянии, а $ENABLED_F(SF_i^j)$ – множество разрешенных переходов j -го базисного ФБ (F_j) приложения A в i -м состоянии.

Приложение функционирует, переходя из одного состояния в другое (начиная от начального) под воздействием срабатывания переходов диаграммы *ECC*.

$$SA_0 \xrightarrow{t_p^k} SA_1 \xrightarrow{t_q^l} \dots \xrightarrow{t_s^b} SA_m$$

Пусть сработал разрешенный переход $t_k^j \in ENABLED_F(SF_i^j)$. Если после его срабатывания $ENABLED_F(SF_{i+1}^j) \neq \emptyset$, то для последующего срабатывания выбирается переход $t \in ENABLED_F(SF_{i+1}^j)$, иначе для срабатывания выбирается переход $t \in ENABLED_A(SA_{i+1})$. Иными словами, используется групповое правило срабатывания переходов: последовательно срабатывают все разрешенные переходы, входящие в один и тот же ФБ. При этом порядок срабатывания переходов в группе не определен

(также, как и в сетях Петри). Возможно наличие нескольких разрешенных переходов, выходящих из одного состояния диаграммы *ЕСС*. Разрешенные переходы в одной группе могут находиться в состоянии конфликта. Два перехода называются конфликтными, если срабатывание одного из них приводит к деактивации другого. Конфликт может возникнуть из-за общего исходного состояния, из-за общих входных событийных переменных, а также внутренних и выходных переменных. При проектировании реальных управляющих приложений опасные конфликты должны быть устранены на этапе проектирования.

Заключение

Предложенные в данной работе формализованное описание и модель систем ФБ использовались при построении: 1) системы верификации, состоящей из конвертора *XML*-представления ФБ в набор продукционных правил, описывающих функционирование системы, и интерпретатора правил на языке Пролог [9]; 2) поисковых и трансформационных систем, предназначенных для использования в системах хранения и синтеза компонентов и проектов на основе ФБ [10]; 3) системы имитационного моделирования на основе ФБ. Направлением дальнейших исследований является использование альтернативных математических аппаратов для построения формальных семантик и моделей ФБ.

Литература

1. Function blocks for industrial-process measurement and control systems - Part 1: Architecture, International Electrotechnical Commission, Geneva, 2005
2. International Standard IEC 1131-3, Programmable Controllers - Part 3, International Electrotechnical Commission, 1993, Geneva, Switzerland
3. Vyatkin V., Hanisch H.-M. A modeling approach for verification of IEC1499 function blocks using Net Condition/Event Systems // Proc. IEEE conference on Emerging Technologies in Factory Automation (ETFA'99), Barcelona, Spain, 1999, pp. 261-270
4. Faure J.M., Lesage J.J., Schnakenbourg C. Towards IEC 61499 function blocks diagrams verification // IEEE Int. Conference on Systems, Man and Cybernetics (SMC02), October 6-9, Hammamet, Tunisia, 2002
5. Stanica P., Gueguen H. Using Timed Automata for the Verification of IEC 61499 Applications // IFAC Workshop on Discrete Event Systems (WODES'04), Reims, France, 2004
6. Thramboulidis K.C. Using UML in Control and Automation: A Model Driven Approach // 2nd International Conference on Industrial Informatics (INDIN'04), Berlin, Germany, 2004
7. Dubinin V., Vyatkin V., Pfeiffer T. Engineering of Validatable Automation Systems Based on an Extension of UML Combined With Function Blocks of IEC 61499 // IEEE International Conference on Robotics and Automation (ICRA'05), Barcelona, Spain, 2005, pp.4007-4012
8. The precise UML group. <http://www.cs.york.ac.uk/puml/>
9. Дубинин В.Н. Одноуровневое представление систем функциональных блоков // Сб.тр. Международной научно-технической конференции “Современные информационные технологии-2005”, Пенза, 2005, с.59-63
10. Дубинин В.Н. Иерархическая структуризация сетевых моделей подсистем ввода-вывода ЭВМ // Вычислительная техника в автоматизир. системах контроля и управления: Межвуз. сб. науч. тр. - Пенза: ППИ, 1985. - Вып.15. - С.33-39
Электронный вариант <http://alice.stup.ac.ru/~dvn/pubs/paper/vt85/index.htm>

11. Дубинин В.Н., Вяткин В.В. Моделирование и верификация приложений IES 61499 с использованием языка Prolog // Материалы VI Международной конференции “Компьютерное моделирование 2005”, С.-Петербург, 2005 (в печати)

12. Дубинин В.Н., Вяткин В.В. Построение поисково-трансформационных систем для поддержки проектирования компонентно-базированных систем промышленной автоматизации // Труды Международных научно-технических конференций “Интеллектуальные системы (IEEE AIS’05)” и “Интеллектуальные САПР (CAD-2005)”, Дивноморское, Россия, 2005 (в печати)