В.Е. ЗЮБИН
(ИАиЭ СО РАН)

Процесс-ориентированный подход к программированию управляющих алгоритмов в среде LabVIEW

В статье предложен вариант реализации управляющих алгоритмов в процесс-ориентированном стиле базовыми средствами LabVIEW. Обсуждаются отличия предлагаемого подхода к созданию сложных алгоритмов управления от существующих событийно-управляемых стратегий.

Ключевые слова: *процесс-ориентированное программирование, сложные алгоритмы управления, LabVIEW.*

V.E. ZYUBIN

Process-Oriented Pattern for Implementation of Control Algorithms in Labview

The paper presents a pattern for process-oriented programming by means of the core Lab-VIEW facilities. The differences of the approach comparing to the known event-driven strate-gies for complex control application programming are discussed.

Keywords: *process-oriented programming, complex control algorithms, LabVIEW.*

Введение

Повсеместное распространение цифровых систем управления и ПЛК, обусловленное экономическими выгодами от их использования, сопряжено с постоянным ростом функциональности и сложности ПО, используемого в задачах промышленной автоматизации. Постоянно растет и стоимость ПО, на разработку которого может приходиться большая часть бюджета проекта. Это обстоятельство мотивирует активность исследователей в области разработки новых подходов, обеспечивающих сокращение временных и финансовых затрат на создание надежных управляющих алгоритмов повышенной сложности. В последние десять лет некоторые из появившихся подходов стали настолько широко использоваться в практике промышленной автоматизации, что начинают ощутимо теснить языки МЭК 61131-3 – наиболее распространенные языки описания управляющих алгоритмов в настоящее время. Давление на языки МЭК 61131-3 наблюдается с двух сторон:

- программными средствами описания сложных алгоритмов управления, концептуально родственными модели конечного автомата. Например, иерархическая машина состояний (*hierarchical state machine*), процесс-ориентированное программирование, так называемая, *switch*-технология, квантовое программирование (*quantum programming*);
- комплексными программными средствами, предоставляющими пользователю не только возможность описания алгоритмов управления, но и средства создания интерфейса оператора. Наиболее популярное средство этого класса – *LabVIEW* компании *National Instruments*.

В случае *LabVIEW* имеется интересное обстоятельство. Базовый язык *LabVIEW G* – язык, построенный на концепции потока данных (*dataflow*), – вызывает затруднения при описании сложных алгоритмов управления, но, с другой стороны, *LabVIEW* имеет развитые средства интеграции сторонних расширений в основной пакет разработки.

В настоящее время существует целая гамма расширений для *LabVIEW*, ориентированных на создание сложных алгоритмов управления, но выбор наиболее эффективного варианта вызывает проблемы у программиста, поскольку не всегда очевиден.

В статье перечисляются распространенные подходы к созданию управляющих алгоритмов и приводятся отличительные особенности процесс-ориентированного стиля программирования. Затем сравниваются механизмы внедрения в *LabVIEW* алгоритмических блоков, реализующих управляющие алгоритмы повышенной сложности, и описываются их достоинства и недостатки. В заключение предлагается способ программирования в процесс-ориентированном стиле базовыми средствами *LabVIEW*.

Обзор подходов к созданию управляющих алгоритмов

На современном этапе развития науки и техники подавляющее число систем управления в области промышленной автоматизации – это цифровые системы управления. Ядро этих систем – программируемый логический контроллер (ПЛК), содержащий цифровое устройство на основе микрокомпьютера, модули ввода-вывода физических сигналов для связи с объектом управления и ПО, реализующее заданный алгоритм управления и вспомогательные функции: арифметические вычисления, логическое управление, регулирование параметров, обслуживание таймеров/счетчиков событий, связь и передачу данных, отображение текущего состояния и др.

Для программирования ПЛК необходимы некоторый язык и стратегия программирования, основанные не только на математическом формализме в виде модели алгоритма управления, но также и на понятийном аппарате и терминологии, которые отражают специфику решаемой задачи.

В последние тридцать лет исследователями было разработано большое число формализмов, ориентированных на создание управляющих алгоритмов.

Не претендуя воспроизвести полный список, в качестве примеров можно упомянуть: модель взаимодействующих последовательных процессов [1], диаграммы состояний [2], язык *Esterel* [3], исчисление общающихся систем и его расширения [4–6]. Каждый из формализмов сориентирован на специфическую область: реагирующие (*reactive*) системы, встраиваемые системы, логическое управление, параллельные вычисления, так называемые, "*системы реального времени*", робототехнику и так далее. Из-за обилия моделей практикующему специалисту-автоматизатору очень тяжело определить, какую же модель следует использовать для решения текущей задачи. В результате победил весьма дискуссионный вариант – набор из пяти языков МЭК 61131-3, который существенно упрощает проблему выбора за счет сокращения числа возможных вариантов. Эксперты в области автоматизации критикуют стандарт за его низкую эффективность: слабую выразительность языков, их неоднородность, серьезные ограничения на сложность создаваемых алгоритмов и даже упоминают стандарт в качестве примера того, "чего не надо делать" при стандартизации [7].

Общая неудовлетворенность и насущная необходимость в сокращении накладных расходов на создание алгоритмов управления побуждает исследователей искать новые подходы, альтернативные МЭК 61131-3. Исследования ведутся в трех основных направлениях:

- разработка стилей программирования на языках третьего поколения (языки общего назначения), в основном на Си и Си++ [8, 9];
- создание языков четвертого поколения для программирования ПЛК [10–12];
- событийно-управляемые расширения для пакетов, ориентированных на использования в смежных с ПЛК областях (наиболее известный пример – это пакет *LabVIEW* компании *National Instruments* [13]).

Замечательная особенность этих исследований в том, что во всех случаях используются модели, связанные с моделью конечного автомата (МКА).

Процесс-ориентированное программирование

Популярность МКА обусловлена адекватностью ее использования при создании событийно-управляемых алгоритмов. Классическая МКА имеет массу привлекательных свойств, упрощающих описание управляющих алгоритмов, как-то:

- активный обмен входными/выходными сигналами с некоторой внешней средой (в задачах автоматизации роль такой среды играет объект управления);
- событийно-управляемое изменение алгоритма, часто называемое *поведением*;
- циклическое функционирование, предполагающее исполнение неопределенно долгое время.

Хотя эти свойства прямо не декларируются в самой МКА [14], это не умаляет их полезности и необходимости.

Тем не менее, средствами МКА невозможно отразить всю специфику алгоритмов управления. МКА не поддерживает:

- операции с временными интервалами;
- массовый логический параллелизм, с возможностью порождать и в последующем гибко контролировать исполнение независимых потоков управления;
- структурную (в том числе, иерархическую) организацию алгоритма [15–17].

Причина этих недостатков – исторические условия создания модели [18]. МКА разрабатывалась в первую очередь для формализации создания физических устройств (слово "автомат" в названии модели отражает именно это обстоятельство) и предполагало пять этапов его построения. Первый этап – этап *блочного синтеза* – предполагал некоторую неформализованную операцию по декомпозиции алгоритма на автоматы. Второй этап – этап *абстрактного синтеза* – отводился для собственно описания автомата без привязки к реализующим элементам. А на этапах с третьего по пятый (этапы *структурного синтеза*, *комбинационного синтеза* и *этап реализации)* решались чисто утилитарные задачи построения автомата из элементарны микросхем типа "И-НЕ" и триггеров. Причем задача блочного синтеза так и не была формализована в рамках МКА и выпала из рассмотрения. Даже в современных работах, посвященных МКА, проблема блочного синтеза – проблема разбиения алгоритма на модули (!), – как правило, даже не упоминается.

Программная реализация автомата не соответствует пятиэтапной методике: последние три этапа отсутствуют (так как отсутствуют микросхемы), а реализационные задачи легко автоматизируются.

Недостатки МКА проявляются уже на концептуально-терминологическом уровне, например, термины "входной алфавит" и "выходной алфавит" устарели и перестали использоваться в практике ИТ более 40 лет назад. Выразительные средства МКА находятся на таком низком уровне, что не предполагают даже сравнения чисел и арифметические операции с ними. Эти атавизмы усложняют изучение МКА современным выпускником школы, живущим в условиях тотальной компьютеризации, особенно активно проникающей в образование. Как следствие мы имеем низкий уровень понимания МКА выпускниками вузов и неоправданно частый отказ от использования этой чрезвычайно эффективной концепции при создании управляющих алгоритмов [19]. Скрытый в МКА потенциал заставляет исследователей модернизировать МКА, приспособить ее к требованиям задач автоматизации, современным тенденциям в информационных технологиях и создавать языки на основе таких модернизированных моделей.

Одна из таких ревизий – модель гиперпроцесса (МГ). МГ – математическая модель, имеющая корни в МКА и ориентированная на описание и анализ сложных систем управления [17]. МГ предполагает описание алгоритма управления в виде множества параллельно

исполняемых *процессов*, активно взаимодействующих друг с другом, то есть на формальном уровне решает проблему блочного синтеза. Процесс, – базовое понятие МГ, – представляет собой радикально модернизированный конечный автомат. Процесс задается набором альтернативных функций (в программистском смысле), или, другими словами, процесс – это полиморфная функция. В отличие от полиморфизма, используемого в объектно-ориентированном программировании, полиморфизм процесса событийно-управляемый. В отдельно взятый момент исполнения программы процесс представлен лишь одной из своих функций – *текущей функцией*. Текущая функция может переключаться по событиям. В качестве событий, приводящих к переключению текущей функции, могут выступать: тайм-ауты, изменения переменных и изменения текущих функций процессов гипераавтомата. Текущая функция задает, как следует реагировать на события (изменения во входных данных). Возможными реакциями может быть не только изменение выходных данных (управляющие сигналы для объекта управления), но и управление другими процессами – их запуск и останов. Среди функций, задаваемых при описании процесса, выделяется *начальная функция* – функция, которая становится текущей при запуске процесса. В дополнение к функциям, задаваемым пользователем, в наборе функций процесса есть две *пассивные функции*: функции СТОП и ОШИБКА, в которых процесс не реагирует ни на какие события. Останов процесса означает смену его текущей функции на одну из пассивных. Назначение пассивных функций – организация взаимодействия между процессами через унифицированный механизм их запуска и средства контроля их завершения: успешного или неудачного (например, при обнаружении отказа оборудования процесс может просигнализировать о неудаче, установив в качестве своей текущей функции пассивную функцию ОШИБКА, что может быть проконтролировано запустившим его процессом).

Механизм пассивных функций – гибкое и простое средство для структуризации параллельно исполняемых процессов. Механизм тайм-аутов (который реализован через индивидуальный для каждого процесса счетчик времени, автоматически обнуляемый при смене текущей функции) позволяет генерировать временные события и, тем самым, обеспечивает синхронизм алгоритма управления.

Язык *Рефлекс*, выполненный как диалект языка *Си*, базируется на МГ. Синтаксически и семантически язык сориентирован на описание управляющих алгоритмов в виде множества взаимодействующих параллельных процессов.

Практическое использование языка *Рефлекс* в серии проектов показало:

- органичное и иерархически глубокое разложение управляющих алгоритмов на параллельные процессы (в реальных задачах число достигало тысячи процессов),
- возможность создания программных иерархий, соответствующих технологическому описанию объекта управления структурно, что обеспечивало, в частности, и терминологическую близость описания алгоритма технологическому регламенту;
- широкий набор специфических стратегий и приемов, не свойственных другим стилям программирования.

Перечисленное позволило выделить стиль программирования на языке *Рефлекс* в особый класс, названный *процесс-ориентированное программирование* (ПОП).

Концепт процесса, выполняющий роль основной структурной единицы в ПОП, обеспечивает создание иерархически организованных управляющих программ, естественным образом отражающих событийность, синхронизм и параллелизм, присущие управляющим алгоритмам. Структурное соответствие создаваемых программ целевой технологии (технологическим процессам и операциям) способствует простоте разработки управляющих программ и их дальнейшего сопровождения. Терминологическая адекватность ПОП современным тенденциям в ИТ-индустрии облегчает его изучение.

При исследовании области применимости и свойств ПОП были получены следующие результаты:

- Создан язык *Рефлекс* (также известный как "*Си с процессами*") [12], ориентированный на описание управляющих алгоритмов. В рамках проекта показана возможность семантико-языковой реализации ПОП с использованием Си-подобного синтаксиса.
- Разработана и реализована система управления для установок по выращиванию крупногабаритных монокристаллов кремния методом *Чохральского* [20]. Технологический процесс выращивания монокристаллов чрезвычайно капризен и сложен из-за того, что ведется на фазовом переходе кристалл-расплав. Он многопараметричен по управлению, многокритериален по требованиям к конечному продукту, характеризуется отсутствием масштабной инвариантностью и изменчивостью законов регулирования по мере роста кристалла. Кроме контуров регулирования термосистемы и многокоординатной системы перемещения ТП предполагает нетривиальное логическое управление газовакуумной системой и системой охлаждения, особенно в случае возникновения аварийных ситуаций, чреватых самыми серьезными последствиями. Проект реализован на языке *Рефлекс*. Созданный алгоритм представляет примерно 900 процессов. Максимальная задержка системы на внешнее событие может быть снижена до 10 мс (дальнейшее снижение ограничено модулями ввода/вывода). Показана эффективность применения ПОП при автоматизации сложных, неоднородных и критических производствах.
- Было предложено процесс-ориентированное расширение языка *LISP – Common Lisp Process System* (CLIPS), ориентированное на системы массового обслуживания [21]. Отличительные особенности CLIPS – возможность передачи числовых параметров при запуске процесса и плавающий период запуска процессов, обеспечивающий полную утилизацию производительности вычислительной платформы. В дополнение

к уже существующему объектно-ориентированному расширению *LISP* (CLOS) показана возможность и процесс-ориентированного расширения функционального языка *LISP*.

• Было предложено процесс-ориентированное расширение языка *ST* из набора МЭК 61131-3 [22]. В синтаксис языка *ST* были введены средства описания процессов, их взаимодействия и операции с временными интервалами. Введенные в *ST* механизмы обеспечивают возможность описания управляющих алгоритмов широкого класса без практикуемого привлечения остальных языков МЭК 61131-3 (а именно, *SFC*, *IL*, *LD*, и *FBD*). Продемонстрирована возможность расширения МЭК 61131-3 в сторону ПОП.

• Предложена концепция виртуальных лабораторных стендов (ВЛС) для обучения студентов программированию алгоритмов управления. ВЛС включает в себя виртуальный объект управления (визуализированную компьютерную модель) и возможность создания алгоритмов управления на языке *Рефлекс*. ВЛС создаются на базе пакета *LabVIEW*. Запись алгоритма на языке *Рефлекс* транслируется для встраивания в ВЛС либо через механизм узла формул (*formula node*) в Си-подобный язык [23], либо подается на вход интерпретатору языка *Питон* (в этом случае производится трансляция из *Рефлекса* в *Питон*) [24]. Подход обеспечивает качественное повышение уровня подготовки студента за счет расширения набора практически решаемых задач, введения в обучение игровой составляющей и сокращения рутинной работы с физическими имитаторами. Предложен способ бесшовной интеграции алгоритмических блоков, описываемых на языке Рефлекс, в среду *LabVIEW* через интерпретатор *Питон*.

• Предложена методика итерационной разработки алгоритмов управления на основе использования программных имитаторов объектов управления [25]. Показано, что в силу комплиментарности с управляющим алгоритмом компьютерная модель объекта управления также может быть описана на языке Рефлекс. Основные преимущества подхода – радикальное снижение вероятности логической ошибки в алгоритме и сокращение сроков ввода системы управления в эксплуатацию. При использовании сторонних пакетов (например, уже упоминавшегося пакета *LabVIEW*) компьютерная модель также может включать и визуальные эффекты.

Пакет *LabVIEW* привлекателен *WYSIWYG*-средствами создания пользовательского интерфейса и развитыми возможностями интеграции стороннего оборудования. Во многих случаях эти возможности компенсируют неэффективность языка *G*, используемого в *LabVIEW* для описания алгоритмов. Дополнительный импульс распространению *LabVIEW* в задачах автоматизации придает тенденция все более частого применения в задачах автоматизации x86-совместимых платформ и операционных систем *Windows* [26]. Использование платформы “общего назначения” при разработке и на целевом компьютере означает существенные плюсы: снижение квалификационных требований к программистам и обслуживающему персоналу, сокращение трудоемкости создания управляющих систем и снижение стоимости их сопровождения.

Идея расширить набор базовых средств *LabVIEW* средствами описания управляющих алгоритмов не нова и на эту тему регулярно появляются публикации. Попытки вдохновляются тем, что управляющие алгоритмы сложно описать в терминах языка *G*, а равно, создать их с помощью *FSM*-модуля *LabVIEW*, реализующего классическую МКА.

LabVIEW: событийно-управляемые расширения для сложных алгоритмов управления

На практике используется несколько подходов внедрения в *LabVIEW* алгоритмических блоков с событийно-управляемой структурой:

• бесшовная интеграция кода в *LabVIEW* через недокументированные механизмы [15];

• ручная интеграция кода в *LabVIEW* через механизм *Formula Node Express VI* [23, 27, 28];

• автоматическая интеграция кода через механизм динамически-подключаемых библиотек (*dll*-механизм);

• бесшовная интеграция кода в *LabVIEW* через сторонние интерпретаторы [24].

В сравнительной таблице (см. табл.) приведены основные характеристики подходов.

В случае управляющих алгоритмов высокой степени сложности разработчик имеет возможность выбрать то расширение *LabVIEW*, которое соответствовало бы специфике проекта в наибольшей степени. Однако, из таблицы также видно и одно неприятное обстоятельство: использование сторонних расширений *LabVIEW* и программные прослойки при обмене данными предполагаются даже тогда, когда событийно-управляемая часть алгоритма характеризуется незначительным или средним уровнем сложности. В связи с этим было бы полезно иметь под рукой методику, исключающую использование сторонних расширений, особенно, когда проект предполагает активное использование библиотечных функций *LabVIEW* и развитой интерфейс оператора.

Процесс-ориентированное программирование на языке G LabVIEW

В основу предлагаемой методики, позволяющей преодолеть ограничения классической МКА без использования сторонних расширений *LabVIEW*, положен процесс-ориентированный подход, предполагающий реализацию событийно-управляемой части программы в виде взаимодействующих процессов. Перед тем, как подробно описать эту методику, рассмотрим классическую реализацию МКА в *LabVIEW*.

Как описано в руководствах по *LabVIEW* от *National Instruments* [29], реализацию конечного автомата рекомендуется вести в терминах состояния (*state*). При этом предполагается, что каждое состояние выполняет

Таблица

Событийно-управляемые расширения LabVIEW и их характеристики

Название	Примеры	Достоинства	Недостатки
Встраивание через внутренний механизм *X-nodes*	Пакет *LabHSM*, модули расширения от сторонних производителей (*add-ons*)	– единая область видимости и бесшовное использование переменных в интерфейсной и событийно-управляемой частях программы; – диагностика ошибок; – низкие накладные расходы на организацию исполнения	– использование недокументированных функций *LabVIEW*, чреватое межверсионной несовместимостью; – необходимо использовать среду разработки *LabVIEW* для модификации управляющего алгоритма; – необходимо использовать стороннее коммерческое ПО
Интеграция через *Formula Node Express VI*	Ручная реализация процесс-ориентированного программирования на языке Си	– отсутствие необходимости использовать сторонние пакеты; – низкие накладные расходы на организацию исполнения	– необходимо использовать среду разработки *LabVIEW* для модификации управляющего алгоритма; – обмен данными интерфейсной и событийно-управляемой частями программы через дополнительную прослойку (*data-layer*); – отсутствие контроля ошибок и проблемы качества событийно-управляемой части программы; – ограничения выразительных средств языка *Formula Node*
	Reflex2CFN (транслятор языка *Рефлекс* в Си-подобный язык *Formula Node*), транслятор *Visio2switch*	– диагностика ошибок; – низкие накладные расходы на организацию исполнения	– необходимо использовать среду разработки *LabVIEW* для модификации управляющего алгоритма; – обмен данными интерфейсной и событийно-управляемой частями программы через дополнительную прослойку (*data-layer*); – необходимо использовать стороннее коммерческое ПО (в случае *visio2switch*); – ограничения выразительных средств языка *Formula Node*
Интеграция через *dll*-механизм	Транслятор *Reflex2C* (*Рефлекс* в Си) или ручная реализация событийно-управляемого подхода	– можно изменять событийно-управляемую часть программы без пакета *LabVIEW*; – диагностика ошибок (в случае транслятора *Reflex2C*); – низкие накладные расходы на организацию исполнения	– необходимо использовать стороннее коммерческое ПО для модификаций управляющего алгоритма (например *Vusual C*); – обмен данными интерфейсной и событийно-управляемой частями программы через дополнительную прослойку; – отсутствие контроля ошибок и проблемы качества событийно-управляемой части программы (при ручной реализации на Си)
Интеграция через встраиваемый интерпретатор	Виртуальные лабораторные стенды (основаны на трансляторе с языка *Рефлекс* в *Питон* и *Питон-интерпретаторе*)	– можно изменять событийно-управляемую часть программы без пакета *LabVIEW*; – возможность бесшовной модификации управляющего алгоритма во время исполнения; – диагностика ошибок	– обмен данными интерфейсной и событийно-управляемой частями программы через дополнительную прослойку; – высокие накладные расходы на организацию исполнения (из-за интерпретатора)

некоторые уникальные операции и “вызывает” другие состояния. Перевод диаграммы состояний в запись на языке *G LabVIEW* основывается на приведенной ниже инфраструктуре (рис. 1), где используется:

- цикл по условию (*While loop*), который постоянно исполняет различные состояния;
- структура выбора (*Case structure*), в каждом из своих вариантов содержащая код, который должен исполнять для каждого состояния;
- сдвиговый регистр (*Shift register*), содержащий информацию о переходах в состояние;
- код перехода, определяющий следующее состояние.

Поток диаграммы состояний реализован циклом, в котором отдельные состояния заменены вариантами в структуре выбора. Сдвиговый регистр цикла по условию хранит информацию о текущем состоянии, которая подается на вход структуры выбора.

Используется четыре общих метода, как определить какое будет следующее состояние. Все из них базируются на использовании *перечислителя констант* (*enum constant*) как это показано на рисунке.

Реализация на основе перечислителя констант связана со сложностью модификации. Перечислитель констант определяет вариант в структуре выбора и в случае, когда программист изменяет число состояний конечного автомата, это разрушает связи по всем используемым копиям перечислителя констант. Это обстоятельство вызывает наибольшие трудности при “классической” реализации конечного автомата через перечислитель констант (разумеется, остаются и уже обсуждавшиеся проблемы, касающиеся самой идеи представления алгоритма в виде одного конечного автомата).

Одно из известных решений заключается в использовании строковой переменной для выбора варианта. В этом случае нет необходимости синхронизовать структуру выбора с переменной. Этот метод требует варианта-ловушки – варианта по умолчанию (*'Default' case*), который предполагает выдачу диагностического сообщения пользователю в случае, если строковая переменная была ошибочно инициализирована неописанным значением.

Предлагаемое решение реализации событийно-управляемых частей программы в стиле ПОП продемонстрировано на рисунке 2 (для упрощения показан только механизм взаимодействия между процессами и их организация). Логически параллельное исполнение процессов организовано с помощью тактированного цикла по условию. Реализация процесса основана на структуре выбора и строковой переменной текущей функции процесса (СПТФП), содержащей имя текущей функции процесса (переменные “ПРОЦ Р1” … “ПРОЦ Р6”

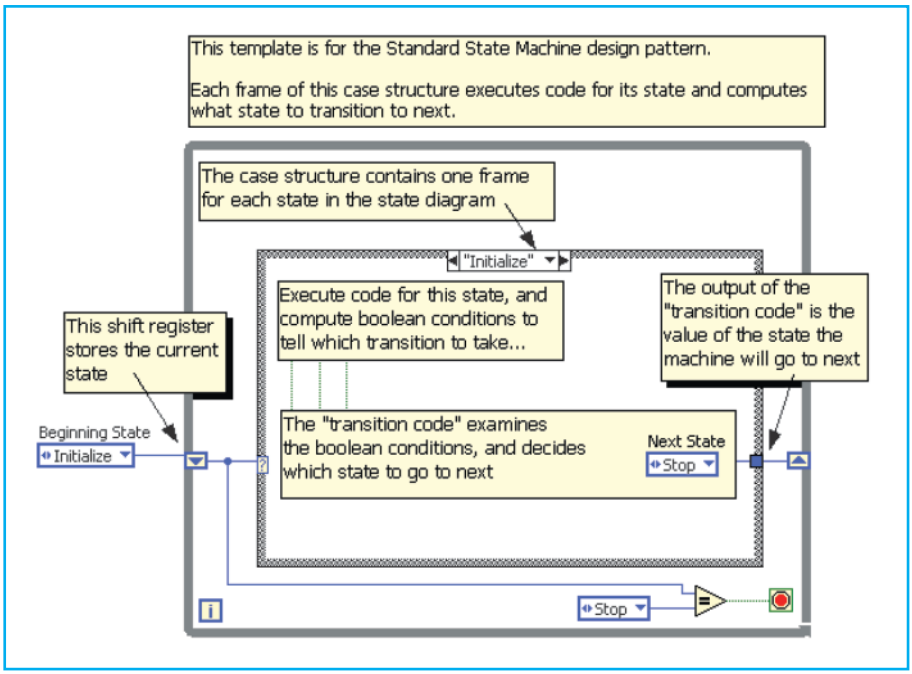

Рис. 1. Шаблон реализации конечного автомата средствами языка G

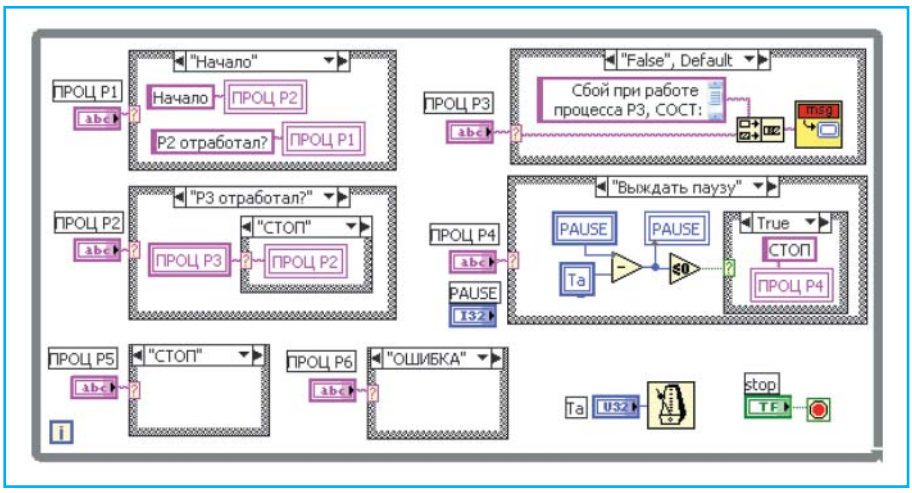

Рис. 2. Предлагаемая реализация процесс-ориентированного стиля программирования средствами языка G

на рисунке 2). Пассивные функции процесса реализованы вариантами с именами “СТОП” и “ОШИБКА” (см. “ПРОЦ P5” и “ПРОЦ P6”). Значение СПТФП по умолчанию – “СТОП”. Для запуска процесса достаточно изменить значение СПТФП (это делается через механизм локальных переменных) на слово “Начало”, которое резервировано для начальной функции процесса (см. “ПРОЦ P1”). Изменение текущей функции происходит при присвоении СПТФП нового значения – имени функции процесса для следующего цикла (см. “ПРОЦ P1”).

Контроль исполнения процесса извне возможен через проверку значения его СПТФП на совпадение с именами “СТОП” или “ОШИБКА” (см. “ПРОЦ P2”). На примере “ПРОЦ P4” (вариант “Выждать паузу”) продемонстрирована возможная реализация синхронизма МГ. Другая допустимая возможность обеспечения синхронизма – использование миллисекундного таймера из набора библиотечных функций *LabVIEW*.

Среди полезных характеристик подхода следует отметить ловушку для неспецированных имен функций: структура выбора, реализующая процесс, содержит вариант по умолчанию, генерирующий диагностическое сообщение оператору об ошибке (см. “ПРОЦ P3”), что облегчает локализацию ошибок.

Еще одно полезное свойство организации процессов через СПТФП – возможность использовать для идентификации функций произвольный язык, что существенно упрощает анализ и модификацию алгоритма русскоязычным программистом.

Из-за низкоуровневой реализации, которая в первую очередь отвлекает программиста от собственно алгоритма управления, в предлагаемом подходе имеются следующие недостатки:

- отсутствует автоматическая диагностика зацикленных и недостижимых функций;
- ограниченный обзор кода (видна лишь одна функция процесса) приводит к низкой читаемости программы.

Среди нюансов реализации была обнаружена сложность организации произвольных иерархических структур (например, при необходимости извне менять текущую функцию процесса с активной на пассивную). Скорее всего, эта проблема обусловлена внутренними деталями синхронизации локальных переменных в *LabVIEW*. Частичное решение проблемы – использование дополнительной логической переменной и структуры выбора, охватывающей деактивируемые процессы.

Обсуждаемая реализация процесс-ориентированного стиля программирования была успешно опробована в проекте по созданию устройства тестирования электронных изделий. Реализованный алгоритм был представлен примерно 50-ю процессами.

Заключение

В настоящее время разработано несколько подходов для реализации сложных алгоритмов управления в среде LabVIEW. Сравнительный анализ этих походов показывает, что существует практическая потребность в методе реализации процесс-ориентированного стиля

программирования на языке G. Такой метод было предложено выстроить на основе использования строковых переменных и структур выбора, заключенных в тактируемый цикл по условию. Частичный контроль корректности создаваемых программ и локализация ошибок выполнен через варианты-ловушки структур выбора.

Отмечены следующие особенности предложенного метода:

- унифицированная организация процессов;
- относительно свободная иерархическая структура алгоритма в процесс-ориентированном стиле;
- единая область видимости и бесшовное использование переменных как в интерфейсной, так и в событийно-управляемой части программы;
- независимость от сторонних пакетов разработки;
- низкие накладные расходы на организацию исполнения;
- возможность использования русскоязычных идентификаторов.

Практическое применение подхода показало надежность и сопровождаемость разрабатываемых управляющих программ, а также снижение трудоемкости их создания.

Список литературы

1. *Хоар Ч.* Взаимодействующие последовательные процессы. / Пер. с англ., М., “Мир”. 1989.
2. *Harel D.* Statecharts: a Visual Formalism for Complex Systems // Science of Computer Programming. Vol. 8, № 3, 1987.
3. *Berry G.* The Foundations of Esterel // Proof, Language and Interaction: Essays in Honour of Robin Milner / *G. Plotkin, C. Stirling, and M. Tofte (eds.)* MIT Press, Foundations of Computing Series. 2000.
4. *Milner R.* Communication and Concurrency // Series in Computer Science. Prentice Hall. 1989.
5. *Kaynar D. K., Lynch N., Segala R., Vaandrager F.* Timed I/O Automata: A Mathematical Framework for Modeling and Analyzing Real-Time Systems // Proc. 24th IEEE International Real-Time Systems Symposium (RTSS’03), IEEE Computer Society Cancun, Mexico. 2003.
6. *IEC 61131-3*. Programmable Controllers. Part 3: Programming Languages. 2nd edn. / International Electrotechnic Commission, 1998.
7. *Crater K. C.* When Technology Standards Become Counterproductive/ControlTechnologyCorporation,1992.URL: http://www.ctc-control.com/customer/elearning/whpapers/ (дата обращения: 19.12.2010)
8. *Samek M.* Practical Statecharts in C/C++: Quantum Programming for Embedded Systems / CMP Books, 2002.
9. *Шалыто А.А., Туккель Н.И.* SWITCH-технология – автоматный подход к созданию программного обеспечения “реактивных” систем // Программирование. № 5, 2001.
10. *Control Technology Corporation*. Quickstep Language and Programming Guide. / Control Technology Corporation, 1996.
11. *Wagner F., Schmuki R., Wagner T., Wolstenholme P.* Modeling Software with Finite State Machines: A Practical Approach / CRC Press, 2006.
12. *Зюбин В.Е.* “Си с процессами”: язык программирования логических контроллеров // Мехатроника. № 12, 2006. URL: *http://reflex-language.narod.ru/articles/* (дата обращения: 12.12.2010).
13. *Блюм П.* LabVIEW. Стиль программирования / Изд-во: ДМК Пресс, 2008.
14. *Кузнецов Б.П.* Психология автоматного программирования // BYTE/Россия. № 11, 2000. URL: http://www.softcraft.ru/design/ap/ap01.shtml (дата обращения: 12.19.2010).
15. *Rumega S.* Using hierarchical state machines in LabVIEW: developing complex event-driven applications using active objects // LabVIEW Technical Resource, № 12 (3), 2005.
16. *Samek M.*, Déjà vu // C/C++ Users journal. June 2003.
17. *Зюбин В.Е.* Язык Рефлекс. Математическая модель алгоритмов управления // Датчики и системы. № 5, 2006.
18. *Глушков В.М.* Синтез цифровых автоматов. М.: Физматгиз, 1963.
19. *Wagner F., Wolstenholme P.* Misunderstandings about State Machines // IEE J. Computing and Control Engineering. Aug, Vol. 16. № 4, 2004.
20. *Зюбин В.Е., Котов В.Н., Котов Н.В. и др.* Базовый модуль, управляющий установкой для выращивания монокристаллов кремния // Датчики и системы. № 12, 2004.
21. *Зюбин В.Е., Хирин Е.* CLIPS – процесс ориентированное программирование на языке LISP // XIV-я международная научная конференция “Современные проблемы информатизации в анализе и синтезе технологических и программно-коммуникационных систем”: Сб. трудов. Вып. 14/ Под ред. д-ра техн. наук, проф. *Кравца О.Я.* – Воронеж: Изд-во “Научная книга”, 2009.
22. *Зюбин В.Е.* Пути расширения языка ST из состава МЭК 61131-3 для задач промышленной автоматизации // Приборы и системы. № 3, 2009.
23. *Зюбин В.Е.* Использование виртуальных объектов для обучения программированию информационно-управляющих систем // Информационные технологии. № 6, 2009.
24. *Зюбин В.Е., Калугин А.А.* Виртуальные лабораторные стенды: обучение программированию задач промышленной автоматизации // Промышленные АСУ и контроллеры № 2, 2009.
25. *Зюбин В.Е.* Итерационная разработка управляющих алгоритмов на основе имитационного моделирования объекта управления // Автоматизация в промышленности. № 11, 2010.
26. *Исаев А.В., Акиншин Л.Г.* ОС Windows в задачах реального времени // Промышленные АСУ и контроллеры. № 3, 2010.
27. *Вавилов К.В., Шалыто А.А.* LabVIEW и SWITCH-технология // Промышленные АСУ и контроллеры. № 6, 2006.
28. *Вавилов К.В.* LabVIEW и SWITCH-технология: методика алгоритмизации и программирования задач логического управления / СПб.: 2005.
29. *Application Design Patterns*: State Machines / NI Developer Zone (Oct 23), 2006. URL: http://zone.ni.com/devzone/cda/tut/p/id/3024 (дата обращения: 12.12.2010).

Владимир Евгеньевич Зюбин – канд. техн. наук, руководитель тематической группы Института автоматики и электрометрии СО РАН, доцент кафедры “Информационно-измерительные системы”, Новосибирского государственного университета
E-mail: zyubin@iae.nsk.su