*УДК 681.51:629.78*

# ИСПОЛЬЗОВАНИЕ ГРАФИЧЕСКИХ ЯЗЫКОВ В ЖИЗНЕННОМ ЦИКЛЕ БОРТОВОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ КОСМИЧЕСКИХ АППАРАТОВ

© 2010 А.А. Калентьев, А.А. Тюгашев

Самарский государственный аэрокосмический университет

Статья посвящена вопросам использования графических языков программирования в жизненном цикле бортового программного обеспечения космических аппаратов. Проведен обзор вариантов применения визуального программирования с использованием графических языков, на стадиях спецификации, проектирования, написания и отладки программ. Подчеркнуто, что главной причиной использования графических средств при создании бортового программного обеспечения космических аппаратов является возможность достижения лучшего взаимопонимания в коллективе специалистов по бортовым системам, алгоритмистов, программистов, что позволяет создавать более надежные и качественные программные системы.

*Графический язык программирования, бортовое программное обеспечение, управляющий алгоритм, космический аппарат*

Современный уровень развития БКУ КА подразумевает использование в качестве центрального компонента БВС, включающего в свой состав одну или несколько БЦВМ [1, 2, 3], связанных соответствующими интерфейсами с бортовыми системами, приборами, агрегатами. При этом непосредственно задачи управления, выполнения КА целевой задачи и реализации для этого требуемой циклограммы работы бортовых систем решаются комплексом БПО, функционирующего в режиме реального времени. Размеры современных комплексов БПО достигают миллионов строк программного кода [3, 4].

При этом спецификой предметной области разработки БПО для КА являются высочайшие требования к надёжности и качеству создаваемых программ управления. Ошибки в управляющих алгоритмах и программах могут иметь весьма тяжёлые последствия, в том числе – потерю дорогостоящих КА, а вместе с ними – срыв решения важнейших научных, народнохозяйственных и оборонных задач, потерю труда многочисленных коллективов, а в случае пилотируемых полётов – и человеческие жертвы.

В сетевом графике работ по созданию КА и ракетно-космических комплексов в целом критическим путём зачастую является именно разработка системы управления. А при создании системы управления, в свою очередь, наиболее трудоёмким и долгим является процесс создания и отладки бортовых программ [3, 4].

При этом в разработку БПО вовлечены сотни людей, включая специалистов по бортовым системам и подсистемам, алгоритмистов-разработчиков УА РВ, программистов, и т.д. Всё названное требует не просто программирования на уровне «искусства» отдельных разработчиков, но использования проработанной технологии программирования, важными чертами которой являются следующие:

- тщательное документирование комплекса БПО;
- как можно большая формализация всех этапов ЖЦ, начиная с постановки задачи;
- создание формальной «базы знаний» о процессах разработки БПО и возможность её использования всеми участниками разработки, снижение зависимости от уникального опыта и

квалификации отдельного исполнителя;
- разделение ответственности.

При этом можно сделать вывод об очень высокой важности создания методов решения следующих задач в применяемых в космической отрасли технологиях программирования:

1. Сокращение сроков и трудоёмкости разработки БПО.
2. Обеспечение требуемого уровня надёжности программ, уменьшение числа ошибок в ПО.
3. Создание удобных и наглядных средств документирования комплексов БПО, упрощающих понимание и взаимодействие участников в процессах его жизненного цикла.

Машинные коды принято считать языками программирования первого поколения.

Повысить производительность труда программиста позволило создание языков программирования второго поколения – языков ассемблера (автокодов). При использовании ассемблера, во-первых, отпадала необходимость запоминать длинные двоичные последовательности – коды команд (вместо чего стали применяться их мнемонические обозначения, например, СУМ, СТОП, и т.д.). Во-вторых, появилась возможность присваивать ячейкам с ключевыми данными, имён (прообраз имен переменных в языках высокого уровня). Все это привело к тому, что программа стала значительно легче восприниматься человеком, повысилось удобство работы, сократились сроки разработки и отладки. Однако теперь стала невозможной прямая загрузка программы в память ЭВМ для исполнения, поскольку для перевода её на язык машинных кодов стала необходимой специальная инструментальная программа – ассемблер.

При использовании ассемблера (автокода), тем не менее, программа должна быть составлена в терминах («на уровне») реализации БЦВМ – её ячеек памяти (с учётом используемых методов адресации), системы команд, уровней прерываний, и пр. Это достаточно далеко от уровня формулировки решаемых на ЭВМ задач.

С целью приближения к уровню абстракции, используемому при постановке и решении задач человеком (математические переменные, формулы, и т.д.), были созданы языки программирования высокого уровня, называемые также языками третьего поколения – Фортран, Кобол, Lisp, APL, Паскаль, Си, Ада, Java и многие другие.

Применение языков программирования высокого уровня позволило в значительной степени за счёт удобства формулировки и записи программ для программиста, повысить качество и надёжность создаваемого ПО, сократить сроки и трудоёмкость разработки. Активно развивается, начиная с конца 1980-х годов, направление ООП в рамках языков высокого уровня. Наиболее известными представителями данного подхода являются такие языки, как Симула, Smalltalk, C++, Java, C#. Как декларируется, ООП позволяет ещё более повысить используемый уровень абстракции в языке программирования и даёт возможность создавать достаточно легко модифицируемые и сопровождаемые большие программные комплексы.

В то же время, уже в середине 1970-х стал очевидным так называемый «кризис» программирования. Его анализу, в частности, посвящена классическая книга Ф. Брукса [5]. Суть сводится к тому, что по сравнению с прогрессом в области аппаратных средств, где за прошедшие с момента появления первых ЭВМ десятилетия быстродействие и объёмы памяти возросли в сотни тысяч раз, прогресс в области разработки ПО не столь впечатляет. Методы обеспечения надёжности аппаратных средств ЭВМ также были в достаточной степени теоретически исследованы и практически отработаны. Методы же обеспечения надежности ПО оставались в основном в

области использования тестов, не гарантирующих на 100% корректность создаваемой программы.

Попытками разрешения кризиса стало появление так называемых «языков программирования четвёртого поколения» (к которым причисляют концептуально достаточно разные подходы) и CASE-средств.

К направлениям исследований и разработок в области языков программирования четвёртого поколения могут быть отнесены:

1. Проблемно-ориентированные языки программирования.
2. Декларативный подход к программированию.
3. Языки визуального (графического) программирования.
4. Использование в программировании подмножеств естественного языка.

Проблемно-ориентированные языки программирования подразумевают, что в языке, наряду с универсальными конструкциями и типами данных, присутствуют встроенные средства для описания понятий, характерных для предметной области, для решения задач которой предназначается данный язык. Примерами могут служить языки, используемые в системах автоматизации управления предприятиями (встроенные языки SAP, 1С и пр.), в которых поддерживаются такие понятия, как бухгалтерский счёт, проводка и пр. Это позволяет значительно короче записывать программы, повысить эффективность.

Декларативный подход к программированию означает, что с программиста снимается обязанность подробного инструктирования ЭВМ, как именно решать задачу (пошагового описания алгоритма). Вместо этого ему необходимо лишь провести постановку задачи некоторым формальным образом, задав существующие ограничения, то есть описать, что требуется получить в качестве результата. Декларативный подход является попыткой воплощения "идеальной" технологии программирования, в качестве которой может быть рассмотрена такая технология, когда по некоторому достаточно неформальному описанию задачи автоматически генерируется синтаксически и семантически корректная программа решения [16]. Поиск решения при этом возлагается на встроенную в систему программирования «машину вывода». Возможно, неплохо описывает ключевую идею декларативного программирования определение Джона Робинсона [27]: программа является теорией, а вычисления представляют собой вывод в этой теории. Ещё одно достоинство декларативных языков - пригодность для формальных рассуждений, которые, в частности, могли бы поднять методы обеспечения надёжности и безошибочности в программировании на новый уровень (известно, что тестирование может помочь обнаружить ошибки в программах, но не может гарантировать на 100% их отсутствия). К сожалению, существующие методы формального анализа программ чрезвычайно трудоёмки. Много усилий было направлено на создание более простых и строгих языков, а также на совершенствование методов анализа. Более того, возникает возможность создания инструментальных средств, позволяющих автоматизировать процессы анализа, преобразования, синтеза программ. Появление таких средств может в корне изменить программирование. При программировании на традиционном (императивном) языке у программиста нет иного выбора, кроме как вникать в низкоуровневые подробности выполнения программ. В декларативных языках, логика и управление отделены друг от друга. Декларативную программу, вообще говоря, проще написать и, что немаловажно, проще понять, чем эквивалентную императивную программу. Помимо этого, возникает возможность передачи ответственности за конкретную реализацию порядка выполнения

операций компьютеру. Более того, появляются возможности использовать нетривиальные алгоритмы планирования вычислений, такие как недерминированные, "ленивые" и параллельные подходы.

Достаточно успешными реализациями декларативного подхода можно считать язык логического программирования Пролог, основанный на выводе в логике предикатов, а также язык запросов к реляционным СУБД SQL.

Идея визуального программирования, также называемого графическим программированием, сводится к тому, что написание программы как текста заменяется в том или ином масштабе её изображением в виде графической диаграммы («рисованием»). В современных системах программирования просматривается тенденция к развитию средств, позволяющих программисту при создании программы оперировать не текстовыми, синтаксическими конструкциями, а графическими образами. Традиционный термин «писать программу» при этом трансформируется в «построить, проектировать программу». Описываться при этом могут самые разные аспекты программного комплекса. Такое богатство возможных средств визуального представления обусловило существование некоторых трудностей при определении понятия «визуальное», или «графическое» программирование. Встречаются варианты: «использование графических средств разработки и визуальной метафоры при создании программного обеспечения» (Microsoft), «программирование, предусматривающее создание приложений с помощью наглядных средств», «графический язык программирования – любой язык программирования, в котором программы задаются путем графического манипулирования элементами взамен текстового манипулирования ими» (Википедия), и пр. Одно из наиболее удачных определений принадлежит Маргарет Барнет, оно используется в энциклопедическом словаре Encyclopedia of Electrical and Electronics Engineering, John Wiley & Sons Inc., New York. Согласно версии Барнет, «визуальное программирование – это программирование, в котором для передачи семантики используется более чем одно измерение».

Визуальное представление обладает рядом значительных преимуществ перед текстовым представлением, в частности – высокой наглядностью и удобством для человека, что позволяет достичь ряда целей, в том числе – сокращения трудоёмкости разработки, повышения качества и надёжности создаваемых программ.

Ещё одним важным направлением в автоматизации программирования, активно развивающимся начиная с 1980-х годов, являются CASE-технологии.

В литературе встречаются различные классификации CASE-средств.

Классификация CASE-средств по уровню [8, 9]:

- верхнего уровня (так называемый upper CASE), средства, которые позволяют построить модель предметной области, рассматривая общую картину, построить генеральный план проекта;
- среднего уровня (middle CASE) – средства поддержки этапов анализа требований и проектирования спецификаций и структуры ПО. Кроме того, CASE среднего уровня обеспечивают возможности быстрой разработки документации;
- нижнего уровня (lower CASE) – средства разработки программ, т.е. генераторы кодов на конкретном языке программирования или конкретной схемы базы данных на языке SQL, и др.

Отметим, что CASE-средствам присущи следующие характерные особенности [10]:

1) наличие развитых графических методов описания и документирования, обеспечивающих удобный и выразительный интерфейс пользователя-разработчика;

2) использование специальным образом организованного хранилища проектных метаданных (репозитория).

При создании языков программирования специалистами было обращено внимание на то, что ответ на вопрос о применимости формального языкового средства должен лежать в сфере психологии, должен быть связан с природой человека и структурой его памяти [11]. Одна из основных проблем, которые встают перед специалистами в области программирования - это неосязаемость предмета деятельности. Живя в материальном мире, человеку довольно трудно и не очень интересно разбираться с «виртуальными информационными объектами», коими по своей сути являются программы. При этом одной из наиболее естественных форм представления (восприятия) информации для человека является графический образ – рисунок, чертеж, схема и т.д. К этой форме человек прибегает всякий раз (возможно неявно для себя), когда необходимо решать (описывать, формулировать) действительно сложные задачи. Человеческая культура «визуально ориентирована». Достаточно упомянуть фотографии, кинофильмы, телевидение, чертежи, чтобы понять, какую важную роль играют изображения в современном обществе. В качестве типичных эпитетов для графического представления используются такие, как "дружественный", "интуитивный", "простой", "привычный", "привлекательный", "понятный", "запоминающийся", "очевидный" и др.

Графика позволяет использовать метафору - представление новых или необычных для пользователя явлений через другие явления, хорошо ему известные из повседневной жизни. Механизм работы метафоры с точки зрения психологии заключается в использовании уже существующих у пользователя знаний - долговременной памяти. Это упрощает освоение нового продукта для новичков в программировании [11]. Примером при создании языкового средства может служить язык релейных схем (ladder diagram) из состава международного стандарта МЭК 61131 [12]. Алгоритм работы устройства на этом языке описывается в виде соединённых реле, которые широко применялись в области автоматизации в 1960-х годах. Язык релейных схем ориентирован на пользователей, знакомых с реле, что упрощает для пользователей переход с реле на микроконтроллеры. Схож по принципу построения и язык функциональных блоков FBD, описанный в том же стандарте. Алгоритм работы некоторого устройства, выраженный средствами FBD, напоминает функциональную схему электронного устройства: элементы типа "логическое И", "логическое ИЛИ" и т.п., и соединения между ними.

При этом к традиционной текстовой форме представления программ может быть предъявлен ряд существенных претензий. В первую очередь, они касаются небольшого выбора доступных выразительных средств, в который входят отступы, комментарии, пустые строки. Несмотря на то, что разработчики систем программирования пытаются обогатить используемые средства представления, (например, современные синтаксически-управляемые редакторы могут автоматически выделять цветом зарезервированные слова языка), для больших программ исходный текст, включающий все аспекты программы вплоть до нюансов реализации, крайне сложен для изучения. Программист, просматривая текст, должен удерживать в уме большое количество деталей [14]. При этом структура (модульность и пр.) и логика текстовой программы, в отличие от визуального представления, при чтении естественным образом не просматриваются.

Можно отметить удачное соответствие графики требованиям этапов постановки задачи и проектирования. Использование визуального представления позволяет

создавать более компактные и наглядные спецификации. Особенно важно, использование графики позволяет упростить общение с заказчиком и, возможно, представить понятия проблемной области. Лучшее понимание задачи позволяет снизить количество ошибок (наиболее трудноустраняемых) в программах, возникающих на первоначальных этапах жизненного цикла – постановке задачи и спецификации требований.

Упрощается также понимание структуры программы в процессе взаимодействия алгоритмистов, различных программистов и других специалистов, вовлечённых в процессы создания ПО. В разработке сложных программных комплексов, таких, как комплекс БПО КА, задействованы сотни участников. Преимущества визуального представления за счёт улучшения координации и взаимопонимания участников работ, дают возможность повышения производительности труда.

Визуальный подход можно отнести к одному из способов автоматизации программирования, который ориентирован на приближение программирования к способу мышления человека. Повышение производительности труда программиста в этом случае связывают с большей наглядностью (понятностью) программ и более комфортными условиями труда, что, в конечном итоге, приводит к повышению надёжности результирующих программных продуктов. В то же время, визуальное программирование расширяет "армию труда" в сфере программирования, поскольку к узкому кругу профессиональных программистов, разбирающихся в деталях традиционных языков, в новых условиях может подключиться достаточно большое количество непрофессионалов, специалистов в других областях, тем не менее, становящихся способными разрабатывать качественные программные продукты с использованием новых визуальных средств.

Неудивительно, что в настоящее время появилось большое количество графических (визуальных) языков программирования.

Графически могут быть представлены самые различные аспекты ПО, на основе чего становится осуществимой классификация подходов в визуальном программировании.

Известна метафора Н. Вирта, согласно которой «Программы = Алгоритмы + Структуры данных». Соответственно, при визуальном описании ПО упор может делаться, с одной стороны, на описание данных, с другой стороны – на описании алгоритмов (control flow).

При этом при графическом описании данных, с одной стороны, упор может быть сделан на описании их статической структуры, в частности, весьма развитым является данное направление в современных БД. На основе методологии DFD существует ряд промышленно используемых CASE-средств, в частности, ERwin, позволяющих по визуальному описанию автоматически генерировать описание структуры БД на языке SQL. Достаточно хорошо известными являются и визуальные конструкторы запросов к СУБД.

С другой стороны, можно сделать упор на том, чтобы описать графически потоки данных (dataflow), т.е. на динамике процесса обработки данных и их передаче от одного программного модуля другому.

В диаграммах типа control flow обычно стрелки соответствуют передаче управления, а в диаграммах dataflow – передаче данных.

В случае упора на описание потока управления (control flow) теоретической основой визуального представления программы служит её управляющий граф. Развитие теории управляющих графов во многом происходило на базе работ, выполненных учёными СССР [18,19,20,21]. Имеется множество практических реализаций визуального подхода к представлению управляющего

графа программы. Классической версией является стандартная блок-схема [23, 24]. Ещё одной альтернативной реализацией описания потока управления в программах являются Р-схемы Вельбицкого [25]. Существенно модифицированным подходом к блок-схемам служат технология ГРАФИТ-ФЛОКС [26] и язык ДРАКОН [26].

Программные комплексы БПО представляют собой яркий пример сложных систем, состоящих из большого числа взаимосвязанных подсистем и отдельных элементов. Этот аспект подразумевает возможность описания в графической форме структуры системы и связей между отдельными компонентами (модулями, в ООП – классами, и пр.). Существует ряд средств, в которых графически представляется декомпозиция программы на модули или классы (ярким примером здесь служит язык UML). При этом связи могут носить различный характер (информационные, управляющие, и т.д.), а компоненты – являться территориально удалёнными и функционирующими на разных ЭВМ (описание распределённых систем и их динамики), что также обогащает диапазон возможных видов графических представлений.

Сложные системы имеют некоторое неэлементарное поведение, в них протекают (возможно, параллельные) разнообразные процессы. Этот (динамический) аспект представления программных комплексов также может быть описан в графическом виде.

Так, для моделирования управляющих систем часто применяемой является концепция состояний, что послужило основой для создания ряда видов графического представления, основанных на визуализации состояний и переходов между ними (язык SFC группы стандартов МЭК 61131 [12], графическая switch-технология, и пр.). При этом теоретической основой здесь могут быть как конечные автоматы, так и сети Петри [22].

При графическом описании систем реального времени существенной становится синхронизация параллельно протекающих процессов, для которых популярно наиболее наглядное представление в виде циклограммы (или диаграммы Ганта). К сожалению, в наиболее распространённых методологиях визуального моделирования, в частности, UML, описание этих важных аспектов до недавнего времени не было поддержано. И только в последней версии UML 2.0 был сделан первый шаг в данном направлении. Применение подобного описания для УА РВ рассмотрено, в частности, авторами настоящей работы в [10]. Представленная там технология ГРАФКОНТ позволяет сочетать преимущества визуального и декларативного подхода к программированию. Фактически описывая (процесс при этом сводится к интерактивному пошаговому графическому конструированию) постановку задачи в виде требуемой циклограммы работы УА РВ, разработчик может автоматически получить готовый текст управляющей программы, реализующей данную циклограмму.

Существование целого набора различных срезов (представлений) ПО в визуальной форме привело к появлению комплексных визуальных методологий, включающих различные виды диаграмм. К примерам таковых, в частности, относятся UML [15] и комплекс стандартов описания микроконтроллеров МЭК 61131 [12]. Отдельные диаграммы (подъязыки) в таком случае могут рассматриваться как принадлежащие к разным классам графических языков программирования.

Некоторые графические языки создаются с исследовательскими целями в рамках научных работ. Можно упомянуть в этой связи также методологию графо-символического программирования (ГСП), развиваемую на факультете информатики СГАУ под руководством А.Н. Коварцева [16].

В отдельный подкласс могут быть выделены также учебные языки, в

противовес используемым в индустрии. Использование графических средств представления в обучении программированию широко распространено, что вполне естественно. Наглядность и удобство графических средств позволяют легче осваивать такие базовые понятия как алгоритм, условие, цикл и пр. Интерактивные инструментальные средства, позволяющие графически конструировать программы и вносить изменения «на лету», весьма популярны в школах и университетах в разных частях планеты. Ярким примером ориентированного на первоначальное обучение программированию графического языка является созданный Аланом Кэем объектно-ориентированный Scratch. В СССР для поддержки школьного курса информатики был разработан «язык пи». При описании CASE-технологий была приведена классификация по уровням. Средства upper CASE позволяют строить модель предметной области, middle CASE предназначены для поддержки этапа проектирования структуры ПО и разработки документации, и так называемые lower CASE включают генераторы кодов (текста программ) на конкретном языке программирования или схемы базы данных на языке SQL. По подобному критерию графические средства поддержки процесса программирования могут быть также разделены на средства (допускающие отклонения от строгой формализации), используемые для лучшего понимания проблемы человеком и документирования, используемые на этапах анализа и проектирования, и средства, позволяющие генерировать управляющую программу или непосредственно графической программе быть исполненной (интерпретированной) некоторой средой выполнения. С точки зрения центрального описываемого аспекта программ в основном учебные языки концентрируются на описании потока управления.

Конечно, в случае рассмотрения языков программирования возможность исполнения программы на ЭВМ является достаточно принципиальной. С этой точки зрения, к собственно «языкам графического программирования» могут быть отнесены средства, позволяющие программе на графическом языке быть исполненной на конкретной ЭВМ, или быть преобразованной в язык автокода (ассемблера) некоторого компьютера или в программу на универсальном языке программирования, которая затем может быть оттранслирована в машинные коды. Естественно, как правило, графическое описание должно в этом случае быть дополнено (уточнено) для придания ему достаточно строгой формализации, допускающей исполнение на ЭВМ. Такого рода дополнительное описание может представляться в виде таблиц (таблицы ФЛОКС системы ГРАФИТ-ФЛОКС), текстов, XML-файлов и т.п.

Для использования на различных стадиях проектирования и разработки БПО КА ДЗЗ графических языков программирования имеется ряд весомых предпосылок, среди которых:

1. Высокая трудоёмкость создания БПО, необходимость повышения производительности труда разработчиков.
2. Крайне высокие требования к надёжности, которые вызывают необходимость всемерного снижения вероятности внесения ошибок в программы.
3. Сложность непротиворечивой спецификации требований к УА РВ, достижения взаимопонимания между проектировщиками бортовых систем и входящей в нее БА, алгоритмистами и программистами.
4. Необходимость тщательного документирования разрабатываемого БПО, адекватного, незамедлительного и точного отражения в документации всех изменений, вносимых в ПО (поддержания актуальности документации).

5. Вероятность использования различных платформ – БЦВМ на борту КА в настоящее время и на перспективу.

Однако, использование существующих средств при разработке БПО КА ДЗЗ не представляется целесообразным в силу ряда причин, среди которых следующие.

Системы визуального конструирования пользовательского интерфейса неприменимы к БПО автоматических КА ДЗЗ ввиду отсутствия в нём взаимодействия с человеком-пользователем.

Большая часть разработок в области визуального программирования носит исследовательский или учебный характер и не может быть использована в промышленных масштабах для решения сложных и ответственных задач большой размерности, к которым относится проектирование БПО КА.

Языки графического описания структур данных практически не имеют значимой области применения при создании БПО КА в силу специфики решаемых на борту задач, не связанных с обработкой сложных структурированных данных, а скорее сводящихся к контролю логических условий, выдаче команд на БА и запуску требуемых функциональных программ из комплекса БПО. Спецификой УА РВ является необходимость чёткой реализации логики управления БА, что требует соблюдение заданной последовательности проверок условий и выдачи необходимых в соответствии с требуемой циклограммой работы КА управляющих воздействий на бортовые системы и программы.

При этом число возможных состояний УА РВ определяется мультипликативно количеством сочетаний возможных состояний БА КА во все системные моменты времени и является практически необозримым. В силу этого, а также отсутствия в большом числе такого рода средств поддержки реального времени, неприменимыми при разработке БПО КА являются рассмотренные графические средства, ориентированные на состояния (SDL, диаграммы конечных автоматов UML, SFC).

Неадекватными проблематике УА РВ являются и графические языки описания потоков данных, к которым, в частности, относятся FBD и LD из стандарта МЭК 61131-3. В них в явном виде отсутствует отражение последовательности и логики выполнения отдельных операций (основным принципом активации выполнения той или иной функции в языках потоков данных является наличие на входе исходных данных).

Не применяется в силу повышенных накладных расходов и требуемых ресурсов при создании БПО КА ДЗЗ объектно-ориентированное программирование, в силу чего отпадает возможность использования объектно-ориентированных визуальных технологий, таких, как диаграммы классов UML, Visual Age или HiAsm.

Наибольший потенциальный интерес представляют графические языки, описывающие поток управления (control flow). Однако, они также недостаточно адекватны задачам проектирования и разработки БПО КА ДЗЗ.

Язык графических блок-схем в чистом виде обладает недостаточными выразительными средствами для отражения особенностей УА РВ для КА, в том числе – особенностей исполнения в реальном времени, и не имеет общепринятой реализации в виде настраиваемого на генерацию программ на произвольных языках инструментального средства.

Технология графосимволического программирования ГРАФ ориентирована

на достаточно узкий круг приложений (Windows-приложения с базовым языком программирования Си). Помимо этого, она не поддерживает приложений реального времени.

Диаграммы деятельностей и активностей UML, с одной стороны, обладают достаточными выразительными возможностями (возможно, даже несколько ими перенасыщены, что приводит к сложности для восприятия и отсутствию концептуальной целостности). С другой стороны распространённые CASE-средства, ориентированные на язык UML, не имеют возможности автоматической генерации программ по данным разновидностям диаграмм. В целом же использование UML при создании БПО КА ДЗЗ вообще затруднено тем, что это – объектно-ориентированная технология.

Весьма неудобным и явно нишевым средством, ориентированным на промышленные контроллеры, является язык потоковых диаграмм FC системы IsaGraf.

Система Algorithm Builder, которую можно признать достаточно удачной, также, к сожалению, ориентирована исключительно на узкий сегмент приложений – программирование микроконтроллеров Atmel.

В настоящее время отсутствуют инструментальные средства поддержки Р-схем программ Вельбицкого для приложений реального времени. Видимо, наиболее близким, в том числе и в силу своего происхождения (разработан в НПЦ АП имени Пилюгина), к задачам создания УА РВ для КА ДЗЗ, является язык ДРАКОН и поддерживающая его технология ГРАФИТ-ФЛОКС. В ней присутствуют и средства поддержки выполнения в реальном времени. Однако, и в ней, во-первых, отсутствуют средства гибкой настройки на генерацию программ на различных ассемблерах или языках высокого уровня. Во-вторых, в принципе не поддерживается понятие входа программы, являющееся ключевым в технологии разработки БПО КА ДЗЗ. И, наконец, в ней в качестве основы применяется графическая нотация блок-схем, недостаточно выразительная, как уже было отмечено, для полноценного отражения особенностей УА РВ.

Таким образом, можно сделать следующие выводы.

1. При создании БПО КА ДЗЗ целесообразно применение графических языков и средств программирования, поддерживаемых интегрированными средами разработки.
2. Существующие графические языки недостаточно адекватны специфике УА РВ для КА и решаемым на различных стадиях ЖЦ БПО проблемам.

Поэтому можно рекомендовать использование при создании БПО КА ДЗЗ специально разработанных графических языков и технологий, в частности – систему ГРАФКОНТ/ГЕОЗ, изначально поддерживающую и дополняющую существующую технологию разработки БПО КА ДЗЗ и в достаточной степени учитывающей их специфику.

**Библиографический список**

1. Соллогуб А.В., Аншаков Г.П., Данилов В.В. Космические аппараты систем зондирования поверхности Земли. Под ред. Д.И. Козлова.- М.:Машиностроение,1993.

2. Управление космическими аппаратами зондирования Земли:Компьютерные технологии / Д.И.Козлов, Г.П. Аншаков, Я.А. Мостовой, А.В. Соллогуб.- М.:Машиностроение,1998.

3. Теоретические основы проектирования информационно-управляющих систем космических аппаратов / В.В. Кульба, Е.А. Микрин, Б.В. Павлов, В.Н. Платонов; под ред. Е.А.

Микрина; Ин-т проблем упр. Им. В.А. Трапезникова РАН.-М.:Наука, 2006.

4. Авиастроение. Том 6 (Итоги науки и техники, ВИНИТИ АН СССР). М., 1978.

5. Брукс Ф. Мистический человеко-месяц или как создаются программные системы: Пер. с англ.-СПб.:Символ-Плюс, 1999.

6. Тьюринг А. Может ли машина мыслить? –М.: Физматгиз, 1950.

7. Terry Winograd, Carlos F. Flores. Understanding Computers and Cognition: A New Foundation for Design. Ablex Pub. Corp., Norwood, NJ, 1986.

8. Калянов Г.Н. CASE. Структурный системный анализ (автоматизация и применение).-М.:Лори, 1996.

9. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем.-М.:Финансы и статистика,1998

10. Калентьев А.А., Тюгашев А.А. ИПИ/CALS технологии в жизненном цикле комплексных программ управления. – Самара:Изд-во Самарского научного цента РАН, 2006.

11. Зюбин В.Е. Графические и текстовые формы спецификации сложных управляющих алгоритмов: непримиримая оппозиция или кооперация? – Сб.трудов VII Международной конференции по электронным публикациям "EL-Pub2002", Новосибирск, 2003.

12. МЭК 65B/373/CD, Committee Draft - МЭК 61131-3. Programmable controllers. Part 3: Programming languages, 2nd Ed. // International Electrotechnic Commission. 1998.

14.Билкун С.Н., Маслюк Г.Ф. О структурном программировании // Программирование. 1976. No 5.

15. Рамбо Дж., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка, 2-е изд. - СПб: Питер, 2007.

16. Коварцев А.Н. Автоматизация разработки и тестирования программных средств. - Самар. гос. аэрокосм. ун-т., Самара, 1999.

17. Зубинский А. Визуальное программирование/ Компьютерное обозрение, 2005, №14(483), с.58-60.

18. Мартынюк В.В. Выделение цепей в схемах алгоритмов // ЖВМ и МФ.-1961.-Т.1,№1.-С.151-162.

19. Янов Ю.И. О логических схемах алгоритмов // Проблемы кибернетики.-М.:Физматгиз,1958.Вып.1. С.75-127.

20. Лавров С.С. Об экономии памяти в замкнутых операторных схемах // Журнал вычисл.математики и мат.физики.-1961.-Т.1.,№4.-С.687-701.

21. Котов В.Е. Введение в теорию схем программ.-Новосибирск:Наука, 1978.

22. Котов В.Е. Сети Петри.-М.:Наука, 1984.

23. ГОСТ 19.003-80. Схемы алгоритмов и программ. Обозначения условные графические

24. ГОСТ 19.701-90 (ИСО 5807-85). Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения.

25. Вельбицкий И.В., Ковалев А.Л., Лизенко С.Л. Графический интерфейс представления алгоритмов и программ // УСиМ. 1988. №4, С.42-47.

26. Пароджанов В.Д. Как улучшить работу ума. Алгоритмы без программистов. –М.:Дело, 2001.

27. J. A. Robinson : E ditor's Introduction. Journal of Logic Programming Vol.1 1984.

28. Kay, Alan. "The Early History of Smalltalk", in Bergin, Jr., T.J., and R.G. Gibson. History of Programming Languages - II, ACM Press, New York NY, and Addison-Wesley Publ. Co., Reading MA 1996, pp. 511-578

# USAGE OF THE GRAPHICAL LANGUAGES IN LIFECYCLE OF EMBEDDED SOFTWARE FOR SPACECRAFTS

© 2010 A.A. Kalentiev, A.A. Tyugashev

Samara State Aerospace University

The paper contains review of the usage of graphical programming languages (□visual programming□) during specification, design, development and verification of embedded software. The main reason for usage of visual programming is obviousness of the graphical materials for human that allows reaching better understanding in collective work of developers and creating more reliable software with the high quality.

*Graphical programming language, embedded software, control algorithm, spacecraft, real-time software*

## Информация об авторах

**Калентьев Анатолий Алексеевич,** д.т.н., профессор, заведующий кафедрой компьютерных систем Самарского государственного аэрокосмического университета, ank@ssau.ru. Область научных интересов: автоматизация проектирования, CALS-технологии, информационная безопасность.

**Тюгашев Андрей Александрович,** д.т.н., профессор кафедры компьютерных систем Самарского государственного аэрокосмического университета, tua797@mail.ru. Область научных интересов: автоматизация проектирования, синтеза и верификации бортового программного обеспечения космических аппаратов, CASE-технологии.

**Kalentiev Anatoly Alekseevich,** professor, hear of the Computer Systems Department Samara State Aerospace University, e-mail: ank@ssau.ru. Field of interest: CAD, CALS-technologies, information security.

**Tyugashev Andrey Alekxandrovich,** professor of the Computer Systems Department, Samara State Aerospace University, e-mail: tau797@mail.ru. Field of interest: Automation of satellite onboard software design, synthesis and verification, CASE-technologies