УДК 681.58.011.56

А.С. Гристан, В.Г. Сухоребрый

Объектная модель программного обеспечения АСУТП

Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ»

Предложены формализация, уточнение и развитие неформальной методики объектного проектирования программного обеспечения АСУТП. Приведенная объектная модель обладает функциональной полнотой, достаточной для создания объектно-ориентированной среды разработки ПО. По сравнению с технологиями традиционных САПР АСУТП (SCADA и SoftLogic) предлагаемая технология отличается унифицированностью: все проектируемые компоненты – это объекты, соответствующие реальным сущностям. Использование распределяемых управляющих компонент позволяет говорить о новом, более высоком, уровне автоматизации самых трудоемких этапов – построении конфигурации и локальной сети АСУ.

Ключевые слова: САПР АСУ ТП, среда разработки ПО, объектная модель ПО, управляющие компоненты, конфигурация АСУ, объектно-ориентированное программирование.

Введение

Все изложенные здесь рассуждения относятся к распределенным системам управления крупными промышленными объектами. Возможно, они пригодятся и в других областях, относящихся к АСУТП, однако авторы такую цель не преследуют.

В статье [1] изложен подход к проектированию программного обеспечения (ПО) АСУТП, базирующийся на систематическом применении методики объектно-ориентированного программирования (ООП). Вот основные концепции статьи [1]:

- большие управляющие компоненты, которые концентрируют в рамках одного объекта полный контур управления технологическим параметром, включая объекты оборудования, сигналы и программные объекты, и не зависят от конкретной конфигурации АСУ;
- распределение пользователем элементов управляющего компонента по устройствам конкретной конфигурации АСУ;
- «интеллектуальная» среда разработки, обеспечивающая «крупнопанельную сборку» и быструю реконфигурацию АСУ за счет автоматического создания сетевых связей и наполнения их потоками сигналов.

Ниже сделана попытка изложить этот подход более формально. Здесь опущены формулировки общеизвестных терминов ООП, хотя и используется несколько ограниченная объектная модель. Это, например, одиночное наследование, поскольку его визуализация с помощью дерева соответствует общепринятому способу представления структуры сложных понятий в интерфейсе пользователя.

Элементы модели

Конфигурация – множество программных и аппаратных объектов, на котором определены отношение ***включения*** и отношение ***коммуникаций***. Два объекта принадлежат отношению включения, если один из них включает другой. Первый объект называется ***владельцем*** или ***контейнером***, второй – ***компонентом,*** или ***элементом***. Отношение включения разбивает конфигурацию на подмноже-

ства, каждое из которых можно представить деревом. Один объект может входить только в одно дерево, т.е. не может иметь более одного владельца. На содержательном уровне ***конфигурация*** определяет программно-аппаратный комплекс, обеспечивающий полную функциональность АСУ.

Отношение коммуникаций – это множество пар объектов, соединенных ***связями***. Это некоммутативное и нетранзитивное отношение.

Связь – элемент отношения коммуникаций: упорядоченная пара объектов, один из которых называется ***источником***, другой – ***потребителем*** данных.

Сигнал – элемент данных, объект класса *TSignal* или любого его потомка. Данные, передаваемые по связям, суть значения и/или другие атрибуты сигналов. Совокупность сигналов, передаваемых по данной связи, называется ***пакетом***. В программе сигнал представлен глобальной переменной – объектом соответствующего класса.

Устройство – элемент оборудования, объект класса *TDevice* или любого его потомка. В нашей объектной модели к устройствам относятся шкафы, стойки, крейты, микроконтроллеры, платы УСО, датчики, исполнительные механизмы- все, что может служить источником и/или потребителем данных.

Слот – обобщенное посадочное место в контейнере для размещения вложенных объектов. Любое устройство, используемое как контейнер, имеет конечное число слотов. Слоты служат для размещения устройств и сигналов. На программные объекты это понятие не распространяется: модель игнорирует ограничения по таким ресурсам процессора, как память и быстродействие, поскольку на этапе проектирования ПО АСУ их практически невозможно проконтролировать. Слот идентифицируется адресом.

Коммуникационная сеть – ориентированный граф, множество вершин которого представлено субъектами коммуникаций (устройствами), а множество дуг – связями. Коммуникационная сеть отражает схему потоков данных АСУ и не обязательно совпадает с физической локальной сетью: связи могут быть реализованы не только проводами, но и стыками, разъемами и т.п. Например, связи между платами крейта или стойки не являются элементами локальной сети.

Адрес – код, определяющий положение объекта оборудования в коммуникационной сети. Адрес, как предписано стандартом [2], представляет собой цепочку элементов, разделенных точками. Каждый элемент соответствует адресу объекта относительно охватывающего объекта. Множество адресов коммуникационной сети называется ее ***адресным пространством***.

Распределяемый управляющий компонент (РУК) – объект, охватывающий полный контур управления некоторым технологическим параметром: датчик(и) – преобразователь(и) – первичная обработка сигналов – логическое управление – регулятор – преобразователь(и) – исполнительный(е) механизм(ы). (Некоторые из перечисленных элементов могут отсутствовать). В эту цепочку следует также включить атрибуты и методы, отвечающие за отображение, операторское управление и архивирование, иначе линии связи между контроллерами и рабочими станциями придется рисовать и заполнять «вручную».

Особенностью использования этого объекта является возможность распределения его частей по устройствам коммуникационной сети. Среда разработки, имея полную информацию о внутренних связях РУК, способна автоматически генерировать связи коммуникационной сети и передаваемые по ним сигналы.

Тестовый управляющий компонент – РУК, контур которого замкнут моделью объекта управления с целью отладки.

Раздел – максимальный элемент организации программы. Раздел не имеет входов и выходов, он может общаться с другими разделами только через глобальные переменные (в частности, сигналы). Раздел может включать методы и атрибуты объектов, процедуры, функции, переменные и константы. Программа раздела обычно представляется FBD-схемой [2], а ее элементы программируются с помощью любых языков, поддерживаемых средой разработки.

Задача – элемент управления выполнением программы, обеспечивающий периодическое или переключаемое выполнение группы ассоциированных разделов. Трактовка этого понятия и набор атрибутов задачи зависят от платформы. С точки зрения нашей модели задача – это программный объект-контейнер, который содержит разделы. Разделы задачи выполняются последовательно, в соответствии с порядком их размещения в контейнере.

Монолитная программа – полная программа управления АСУ или некоторая законченная ее часть, собранная из программных объектов и моделей с целью отладки.

Классы

Для представления классов используются две связанные между собой формы:

- дерево, на котором показано отношение наследования, а также внутренняя структура каждого класса (атрибуты и методы);
- иерархия схем, отражающих связи между членами класса.

На схеме класса в виде вершин присутствуют все его атрибуты и методы. Параметры любого метода изображаются входами и выходами на левой и правой сторонах блока метода. Однако эти входы на схеме класса остаются не подсоединенными: они будут подсоединены к внешним (по отношению к классу) данным при построении программы, которая использует этот метод как элемент объекта данного класса. Внутренние связи между методами отображаются на схеме класса явно. Эти связи, чтобы не смешивать их с входными и выходными параметрами метода, подключаются к его верхней (входы) и нижней (выходы) сторонам.

Заметим, что на схеме класса любая связь между методами является атрибутом; обратное не верно. Связи между методами через параметры, если таковые существуют, могут быть отображены только на FBD-схеме метода, который использует эти методы.

Любой метод, изображенный в виде блока на схеме класса, может быть развернут в исходное представление на одном из языков программирования, поддерживаемых средой разработки.

Особую категорию методов представляют обработчики событий. В отличие от обычных методов, вызовы которых происходят согласно потоку управления, определяемому FBD-схемой, обработчик события вызывается в момент возникновения ассоциированного события, асинхронно по отношению к общему потоку управления.

События, как указывалось в работе [1], могут происходить во время проектирования АСУ (*design time*) либо во время ее работы (*run time*). Ниже в таблице приведен перечень событий, включенный в нашу модель.

Событие	**Назначение**	**Время срабатывания**
ON ALLOCATE	Действия во время размещения в конфигурации	design time
ON CONDITION	Реакция на истинность произвольного условия	run time
ON CREATE	Действия в момент создания объекта	design time
ON DELETE	Действия в момент удаления объекта	design time
ON ERROR N	Реакция на исключение типа N	run time
ON INSERT	Действия во время вставки в контейнер	design time

События, происходящие во время проектирования, генерируются средой разработки и используются для любых нестандартных действий, специфичных для данного класса. Это может быть контроль соответствия стыкуемых элементов, создание, удаление или присоединение дополнительных элементов и т.п.

События ON CREATE и ON DELETE выполняются средой во время проектирования. Они могут быть использованы для создания и удаления объектов, связанных с рассматриваемым. Действия, выполняемые в ответ на аналогичные события во время выполнения, обычно описываются в конструкторах и деструкторах.

Условие ON CONDITION проверяется после выполнения каждого блока FBD-схемы, на которой размещен обработчик этого события, за исключением самого обработчика.

Событие ON ERROR N вырабатывается операционной системой или программами системного ПО. Они же вызывают всех обработчиков типа N.

Обработчик события программирует пользователь теми же средствами, что и любой другой метод. Для событий с параметрами (ON CONDITION, ON ERROR) пользователь задает значение параметра (условие, тип ошибки), которое отображается на схеме класса. Заметим, что *условие* в качестве операндов может содержать атрибуты (включая атрибуты атрибутов-объектов любого уровня вложенности), константы и функции над ними.

Атрибуты класса (альтернативные термины – свойства или поля) могут иметь предопределенный тип, ранее определенный пользователем производный тип или быть объектом любого ранее определенного или определяемого пользовательского класса. К производным типам относятся массивы, структуры, строки, множества, перечислимые типы и любые их комбинации. Атрибут любого указанного выше типа может иметь начальное значение. В случае простого типа – это одиночное значение, в случае пользовательского типа – составное значение, в случае объекта – значения атрибутов всех вложенных объектов, с учетом их типов (простых, производных или объектов).

Оборудование

Компоненты оборудования присутствуют в нашей модели лишь в той мере, в которой они необходимы для проектирования программного обеспечения распределенной системы управления. В частности, они используются для построения конфигурации АСУ. В состав атрибутов входят лишь те свойства, которые используются программами. При желании можно включить также атрибуты, необходимые для вывода полноценной проектной и эксплуатационной документации.

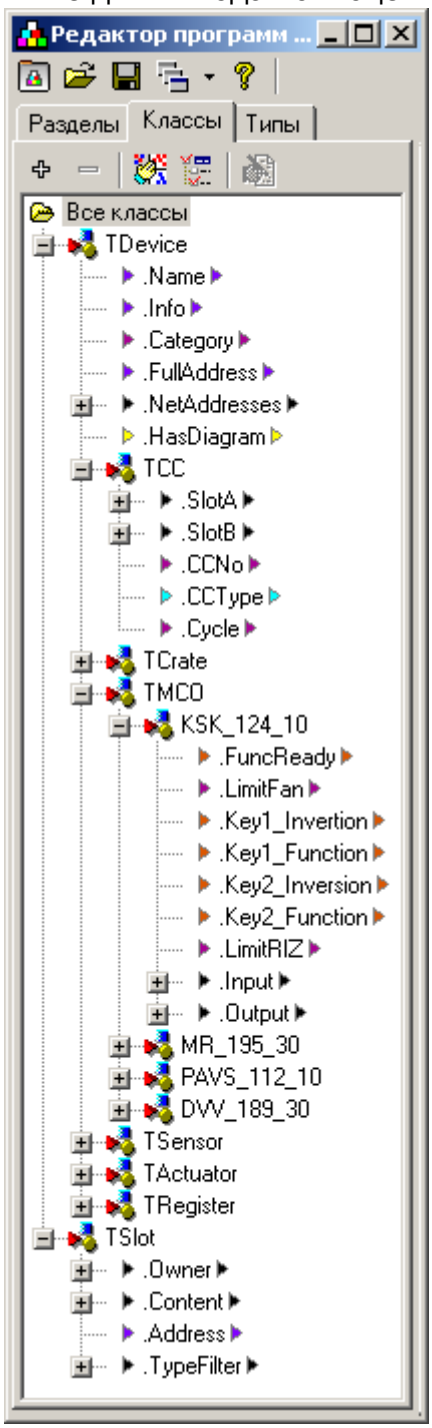


Рис. 1. Классификация оборудования

Возможная классификация оборудования показана на рис. 1. Этот пример основан на типовом проекте АСУ атомной электростанции (АЭС). Для некоторых классов показаны подклассы и атрибуты. Методы не приведены.

Класс *TDevice* является суперклассом всех устройств и содержит их общие атрибуты. Атрибут *HasDiagram* имеет истинное значение для таких категорий устройств, компоненты которых обмениваются сигналами (например, конфигурация, шкаф, крейт). Среда разработки строит схему автоматически, используя информацию о размещении объектов и их внутренних связях. Атрибут *FullAddress* представляет собой цепочку адресов всех адресуемых охватывающих объектов, начиная с самого верхнего уровня и заканчивая адресом данного компонента, разделенных точками, например: 017.B.13.07.

Подкласс шкафов управления *TCC* в качестве атрибутов содержит два слота для вставки крейтов и так называемые паспортные данные. Все остальные свойства шкафа содержаться в полях, наследуемых от класса *TDevice*.

Подкласс модулей связи *TMCO* содержит подклассы конкретных устройств (на рисунке показаны далеко не все типы МСО, применяемые в данной области). В качестве примера возможного набора паспортных данных МСО раскрыт модуль KSK-124-10. В целях компактности здесь также показаны не все атрибуты модуля. Класс *KSK-124-10* имеет два слота (*Input* и *Output*) для «вставки» объектов класса *TRegister*.

Объекты классов датчиков (*TSensor*) и исполнительных механизмов (*TActuator*) представляют собой программные модели источников и потребителей сигналов с определенными свойствами.

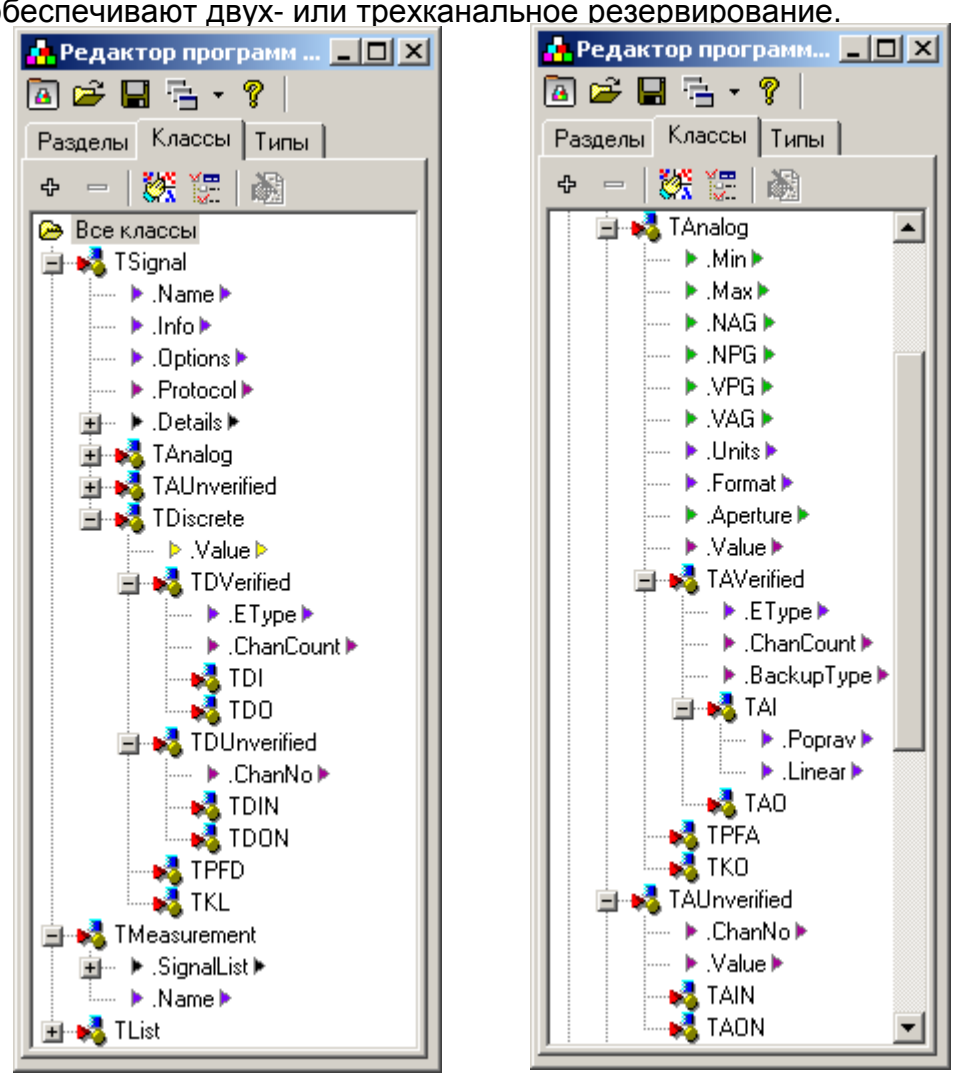
Объекты класса *TSlot* используются в качестве компонентов всеми контейнерами. Атрибут *Owner* ссылается на охватывающее устройство. Атрибут *Content* ссылается на объект, вставленный в слот. Атрибут *Address* – это адрес относительно охватывающего контейнера. Атрибут *TypeFilter* – список

классов объектов, допустимых для вставки. Одним из методов класса *TSlot* может быть обработчик события ON INSERT, который контролирует допустимость вставки, используя атрибут *TypeFilter.*

Некоторые устройства могут не иметь адреса. Они являются фиктивными и вводятся в структуру конфигурации для улучшения наглядности.

Сигналы

На рис. 2 показана возможная классификация сигналов АСУ АЭС. На рис.2, а можно увидеть суперкласс всех сигналов *TSignal*. Его атрибуты используются всеми производными классами сигналов. Атрибут *Details* представляет собой список сигналов, функционально связанных с данными. Например, каждый сигнал класса аналоговых входов *TAI* ассоциируется с группой вспомогательных сигналов различных типов, которые несут дополнительную информацию для программ управления, например: вывод в ремонт, отключение контроля датчика, допустимое рассогласование, значение замещения замера, рабочее значение замера, брак датчика по отклонению, отказ замера, сброс отказа замера. К этой же группе относятся необработанные значения сигнала, которые поступают по разным каналам и обеспечивают двух- или трехканальное резервирование.



а б

Рис. 2. Возможная классификация сигналов

На этом же рисунке раскрыты классы дискретных сигналов *TDiscrete* и резервированного замера *TMeasurement*. На рис. 2, б раскрыты классы аналоговых сигналов *TAnalog* и аналоговых недостоверизованных сигналов *TAUnverified*.

Аналоговые и дискретные сигналы до первичной обработки называются недостоверизованными (классы *TAUnverified* и *TDUnverified*, соответственно). Сигналы, прошедшие первичную обработку (в том числе линеаризацию и достоверизацию), относятся к достоверизованным (классы *TAVerified* и *TDVerified*). Атрибут *Value* для аналоговых сигналов имеет целый тип (используется масштабирование), а для дискретных – булевский.

Класс *TMeasurement* (замер) связан еще с одним способом резервирования: он объединяет сигналы *TAI*, относящиеся к одному технологическому параметру, но измеренные разными датчиками. Ссылки на эти сигналы содержатся в списке *SignalList*.

Следует иметь в виду, что в предлагаемой объектной модели сигналы существуют не сами по себе, а входят в более крупные объекты, которые названы РУК. Поэтому, вопрос о том, в какой класс включить ту или иную группу сигналов, необходимо решать с помощью такого естественного критерия: сигнал должен быть включен в тот класс, методы которого этот сигнал явно используют. Если сигнал используется методами более одного класса, удаляться этот сигнал должен при удалении последнего объекта, использующего данный сигнал.

Поскольку выбрано одиночное наследование (по причине, указанной выше), классификация сигналов не избежала некоторого дублирования свойств. Множественное наследование для такого набора типов было бы рациональнее.

Программы

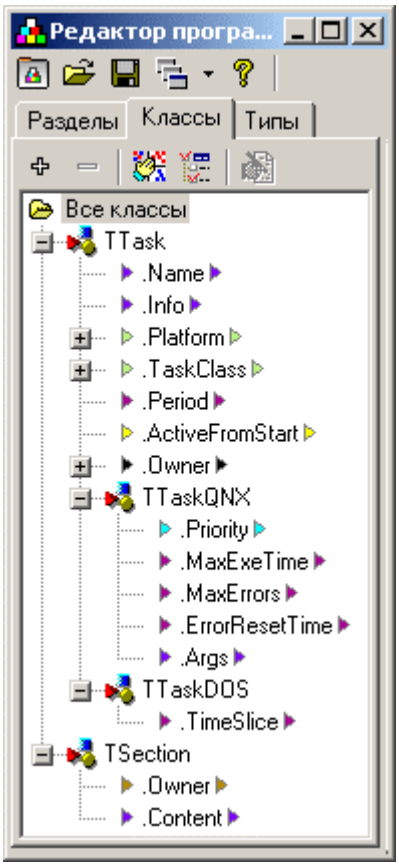


Рис. 3. Классификация программных объектов

Программа, размещенная в одном процессоре, состоит из одного или нескольких разделов (*program* в терминологии стандарта IEC 61131-3 [2]. Один или несколько разделов могут быть объединены в одну задачу. Контроллер может выполнять несколько задач (параллельных процессов) одновременно.

Практически это совпадает с программной моделью, предписанной работой [2]. Отличие заключается в возможности использования в разделах членов (атрибутов и методов) объектов. Более того, несколько разделов могут пользоваться членами одного объекта, причем разделы эти могут принадлежать разным процессам одного или разных устройств коммуникационной сети.

Возможная классификация программ показана на рис. 3.

Объектная модель программ содержит задачи (класс *TTask*) и разделы (класс *TSection*). В обоих классах отсутствуют «слоты», поскольку их количество нашей моделью не ограничивается: соответствие ограничениям по таким ресурсам, как память и время выполнения, может быть проверено лишь на этапе отладки ПО АСУ.

Поскольку набор атрибутов задачи зависит от платформы, класс *TTask* может подразделяться на несколько подклассов, которые соответствуют платформам, поддерживаемым средой разработки. В нашем примере это две платформы: QNX и MS DOS (дополненная средствами разделения времени).

Построение конфигурации

Технология разработки ПО, базирующаяся на предлагаемой объектной модели, значительно отличается от традиционной. Рассмотрим шаги построения конфигурации АСУ.

Шаг 1. Дополнение библиотеки недостающими классами (если таковых не имеется, пропускаем этот шаг). Определяем структуры новых классов, разрабатываем методы, создаем тестовые объекты и отлаживаем их.

Шаг 2. Создание объектов РУК. Создаем и настраиваем необходимые для данного проекта управляющие компоненты. Поскольку сигналы и другие атрибуты РУК относятся к конкретному контуру управления, большинство настроек можно сделать на уровне класса, что сокращает объем настроек объектов.

Шаг 3. Построение монолитной программы. Используя элементы созданных РУК, строим разделы монолитной программы управления и распределяем их по задачам. Проводим отладку монолитной программы на одном процессоре с использованием моделей.

Шаг 4. Разработка конфигурации. Создаем необходимые объекты оборудования и по мере их получения размещаем в них задачи, которые заполняем разделами. Строим программы разделов, используя элементы РУК, полученных на шаге 2. По мере распределения частей РУК по структурным элементам оборудования среда автоматически генерирует связи коммуникационной сети и заполняет их сигналами.

Шаг 3 не является обязательным. Он рассчитан либо на небольшую АСУ, либо на разработку отдельных подсистем. Этот шаг позволяет заранее отладить алгоритмы управления и монолитную программу, что может положительно сказаться на общей трудоемкости разработки ПО.

Выводы

1. Предложены формализация, уточнение и развитие неформальной методики объектного проектирования программного обеспечения АСУТП, представленной в работе [1].
2. Приведенная объектная модель, несмотря на ограниченное количество понятий, обладает функциональной полнотой, достаточной для создания объектно-ориентированной среды разработки ПО.
3. Представленная технология разработки программного обеспечения базируется на систематическом объектном подходе. По сравнению с технологиями традиционных САПР АСУТП (SCADA и SoftLogic) она отличается естествен-

ностью и унифицированностью: все проектируемые компоненты – это объекты, соответствующие реальным сущностям.

4. Использование распределяемых управляющих компонентов позволяет говорить о новом, более высоком, уровне автоматизации самых трудоемких этапов – построения конфигурации и локальной сети АСУ.

Список литературы

1. Сухоребрый, В.Г. Объектное проектирование АСУ ТП / В.Г.Сухоребрый, А.С.Гристан, Ю.К.Швыдкий. // «Мир автоматизации».-2010 - №3. – С. 50 - 54.
2. International standard 1131-3, Part 3: Programming languages, IEC, Division Automatismes Programmables, First edition, 1993.

Рецензент: д-р техн. наук, проф., проф. каф. № 504 В.И. Кортунов, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков

Поступила в редакцию 05.09.12.

Об'єктна модель програмного забезпечення АСУТП

Запропоновано формалізацію, уточнення і розвиток неформальної методики об'єктного проектування програмного забезпечення АСУТП. Наведена об'єктна модель має функціональну повноту, достатню для створення об'єктно-орієнтованого середовища розроблення ПЗ. Порівняно з технологіями традиційних САПР АСУТП (SCADA і SoftLogic) пропонована технологія відрізняється уніфікованістю: всі проектовані компоненти - це об'єкти, які відповідаютьі реальним сутностям. Використання розподілюваних керувальних компонент дозволяє говорити про новий, більш високий рівень автоматизації найбільш трудомістких етапів – побудову конфігурації та локальної мережі АСУ.

Ключові слова: САПР АСУ ТП, середовище розроблення ПЗ, об'єктна модель ПО, клерувальні компоненти, конфігурація АСУ, об'єктно-орієнтоване програмування.

The object model of the software control system

Proposed formalization, refinement and development of the informal methods of object design software control systems. The above object model has a functional fully enoughto create an object-oriented software development environment. Compared to traditional CAD technology control systems (SCADA and SoftLogic) the proposed technology is different commonality: all engineered components - are objects corresponding to real entities. Using a distributed control components suggests a new, higher level of automation of the most labor-intensive steps - build configuration and network control systems.

Keywords: CAD CAM, software development environment, object model, control components, the configuration of ACS, Object-Oriented Programming