Аспектно-ориентированные методы в управлении информационными потоками БД ДП АСУТП

Богданов Н. К.
(АО "АтлантикТрансГазСистема")

В классах БД ДП АСУТП, так же как и в классах программных систем, наблюдается проблема усложнения структуры из-за необходимости поддержки в них различных ограничений и требований к информационной системе в целом. Использование методов аспектно-ориентированного программирования позволяет отделить средства реализации контрактов классов от описываемых ими абстракций сущностей.

Введение. Сущность аспектно-ориентированного программирования

Методы объектно-ориентированного анализа и проектирования позволяют создать модель (архитектуру) информационной системы; провести анализ и создать на его основе модель предметной области. Типизации проектных решений служит широко распространенная концепция шаблонов, эффективные методы анализа и проектирования могут быть оформлены как стратегии.

Однако при разработке программной системы требуется также обеспечить выполнение различных требований к ней. Это могут быть требования к безопасности (необходимость авторизации при проведении транзакций клиент-сервер), качеству обслуживания, синхронизации операций чтения/записи и обеспечению целостности данных и т. д.

Ранее для специфицирования необходимости обеспечения некоторым классом определенных требований было введено понятие контракта [1]. Однако, поддержка любого требования, не относящегося к сущности, описываемой классом, усложняет его структуру. Более того, существует ряд требований, общих для многих различных классов или не являющихся функциональными, реализация которых в отдельных классах исключительно затруднена (такие требования называют «пересекающими» (crosscutting)). Требуется введение некоторого дополнительного программного «слоя», на который было бы возложено выполнение «контрактных обязательств» классов, абстрагирующих сущности предметной области.

Для этого в 1997 г. группой разработчиков из Xerox PARC во главе с Г. Кикзалесом была предложена концепция аспектно-ориентированного программирования (АОП) [2]. Ими было введено понятие аспекта, которым является то свойство системы, которое не может быть явно реализовано в виде процедуры. «Аспекты имеют тенденцию не быть элементами функциональной декомпозиции системы, но скорее быть свойствами, которые системно воздействуют на производительность или семантику компонентов». В этом аспекты противоположны компонентам, «имеющим тенденцию быть единицами функциональной декомпозиции системы». Цель АОП – «поддержать программиста в четком разделении компонентов и аспектов друг от друга, обеспечивая механизмы, которые сделают возможным абстрагировать их и объединять для получения системы в целом». (На русском языке концепции и преимущества АОП описаны в [3]).

Для успешного применения любой техники программирования требуется не только ее поддержка языком программирования, но и возможность формализации принятых решений на этапе проектирования. Из этого следует необходимость наличия как минимум графической нотации для записи моделей, как максимум – методологии, формализующей процесс проектирования, и поддержки новой технологии в CASE-средствах разработки.

Моделирование аспектов на UML

Как отмечают авторы [4], в то время как существует поддерживающий АО – концепции язык программирования AspectJ, отсутствует реализованный язык моделирования, поддерживающий проектирование AspectJ-программ. Предложениям по разработке подобного языка посвящено большое число работ, представленных на различных конференциях в 1998-2002 гг.

Подавляющее большинство исследователей предлагают основываться на стандарте UML и применять существующие в нем механизмы расширения графической нотации сущностей и отношений (стереотипы, ограничения, помеченные значения) для описания дополнительных концепций АО-проектирования.

Так, в работе [5] предлагается ввести три новых концепции: группы (для целей классификации гетерогенных и распределенных сущностей), пересекающие отношения (позволяющие программисту определить «точки пересечения» аспекта с функциональной программой), аспектные классы (реализующие расширения программы в точках пересечения). Графически это предполагает использование имеющихся в UML элементов: классов и ассоциаций с добавлением стереотипов «group», «pointcut»,

«aspect». Для методов аспектного класса вводятся стереотипы «before», «after», «around», описывающие момент их вызова по отношению к вызову «пересекаемых» функций, а также предлагается набор правил определения и интерпретации семантики ролей и кратностей для пересекающих отношений. Далее при составлении диаграмм мы будем придерживаться именно этой нотации.

Цель следующей части статьи - дать представление о применимости АО-методов в обработке информационных потоков в объектно-ориентированной среде, вариантом реализации которой является объектная (объектно-иерархическая) БД программного комплекса ДП АСУТП.

БД ДП АСУТП и задачи управления информационными потоками

Большинство СУБД ДП АСУТП используют иерархические модели набора сущностей, а не распространенные в остальных классах систем хранения данных, реляционные [6]. Это связано с более высоким быстродействием выборки данных, простотой программной реализации, возможностью отразить в иерархии групп данных структуру автоматизируемого производства, наличием методов относительной адресации. При использовании модели набора сущностей область хранения данных организована линейно, но упорядочивание объектов производится с использованием методов наименования, а логические взаимосвязи между ними приводят к построению некоторого графа. Исходя из этого, в данной работе рассматривается класс, так называемых, объектно-иерархических СУБД, предоставляющих механизмы упорядочивания хранимых объектов и программный интерфейс манипулирования ими.

БД ДП АСУТП является как приемником, запрашивающим данные от внешних систем, так и их пассивным источником и, таким образом, выполняет роль маршрутизатора информационных потоков от систем автоматики и телемеханики к графическим приложениям, системам коммерческого учета и планирования производства, экспертным системам. При этом возникают общие для систем хранения и обработки данных задачи: выполнение функциональных операций; поддержание целостности и эквивалентности реплик данных; а также специализированные – взаимодействие с подсистемой информационного обмена и т.п.

В науке каждая новая точка зрения влечет за собой революцию в ее технических терминах

Ф. Энгельс

Реализация функциональных операций

Простейший случай использования аспекта – реализация некоторого функционального требования, необходимого разным (не имеющим общего базового) классам. В этом случае аспект становится похож на статический класс-утилиту (utility class), с более четко формализованным в проекте правилом его использования.

Для примера рассмотрим класс, хранящий некоторое значение и его метку времени, имеющий два полиморфных метода записи нового значения: один с передаваемой меткой времени, другой – без нее. Тогда в случае, когда метка времени не передана, необходимо определить текущее время системы и записать его. Для этих целей введем аспектный класс TimeStamping. Его можно показать на диаграмме классов (рис. 1а), при этом мы показываем связанность аспекта не с одним, а с группой классов, которые объединяет, в данном случае, только необходимость получения значения текущего времени. То, что в принятой нотации группа моделируется как вариант класса, а не пакета UML, позволяет указать для нее методы, подразумевая, что они присутствуют во всех классах, составляющих группу. (Здесь и далее мы задаем имя группы Signal, подчеркивая, что оперируем с множеством классов, хранящих значение некоторого аналогового, дискретного, логического измерения, единицу справочной информации или производную от этих значений величину). Диаграмма взаимодействия (рис. 1б) показывает последовательность выполняемых операций после выполнения связывания (генерации программного кода). При вызове метода SetValue(value) происходит «пересечение»; мы показываем это тем, что на «линии жизни» объекта класса Signal не отмечен фокус управления, который сначала переходит к экземпляру аспектного класса, и только потом возвращается им.

Другой пример связывания некоторого аспекта с каждым из некоторого набора объектов в индивидуальности – необходимость отслеживать факт изменения хранимого значения. Перехват вызова метода записи нового значения позволяет установить флаг наличия изменения. Программе-серверу,

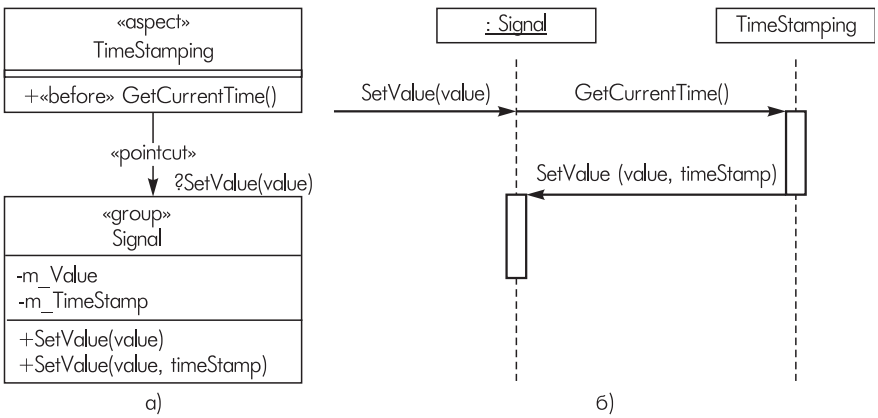

Рис. 1. Реализация функциональных операций аспектным классом.

обрабатывающему запросы внешних систем на получение измененных данных, достаточно будет анализировать состояние данного флага, а не выполнять сравнения хранимых копий предыдущих переданных значений с текущими.

Синхронизация расчетов и изменений

В БД ДП АСУТП часто выполняемой операцией является расчет агрегированных значений, например, определение максимального из множества значения или расчет мгновенного расхода жидкости или газа, используя оперативные данные о давлении, его перепаде на диафрагме и температуре, а также ряд заданных нормативных поправочных коэффициентов. В модели мы можем отобразить это, установив ассоциацию один-ко-многим (часто соответствующей отношению контейнер-элемент) и использовав класс-ассоциацию Multiplexer (в который заложена логика преобразования нескольких исходных значений в одно производное). Объекты-элементы являются источниками данных для своего контейнера, в свою очередь некоторая подсистема опроса систем автоматики является источником оперативных данных для них самих. Эти взаимосвязи можно показать, используя диаграмму классов UML (рис. 2).

В системе, управляемой событиями, при изменении хотя бы одного из значений, участвующих в расчете результата и хранящихся в объектах-элементах, должен происходить пересчет формулы и запись нового расчетного значения в объект-контейнер. Рассмотрим для примера расчет некоторого состояния по двум исходным логическим сигналам «открыт/закрыт» (рис. 3).

В этом алгоритме есть упущение – перерасчет результата происходит при поступлении первого же из изменений. В случае, когда произошло изменение обоих значений, выполнение промежуточного расчета формулы агрегирования с использованием одного нового и одного устаревшего значения избыточно, и, скорее всего, ошибочно. При этом мы не можем заранее утверждать, что всегда при поступлении нового значения одного из сигналов поступает и новое значение другого. Таким образом, мы приходим к требованию отслеживать режим поступления обновлений. Для решения этой задачи можно предложить блокировать источникам данных передачу сообщений об изменении значений в объектах-элементах на период записи всего множества поступивших обновленных значений. Тогда события об изменении поступят в класс-контейнер после записи обеих новых значений и расчет будет выполнен два раза, но с актуальными данными. (В случае уверенности, что при поступлении первого же из извещений актуальны оба значения, можно сократить число перерасчетов до одного).

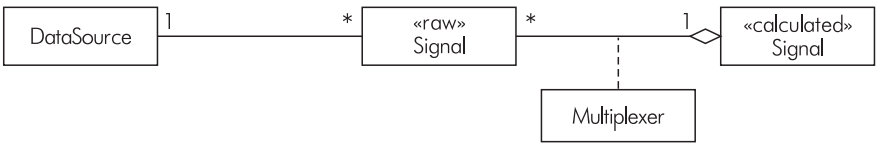

Рис. 2. Типовые отношения классов БД ДП АСУТП.

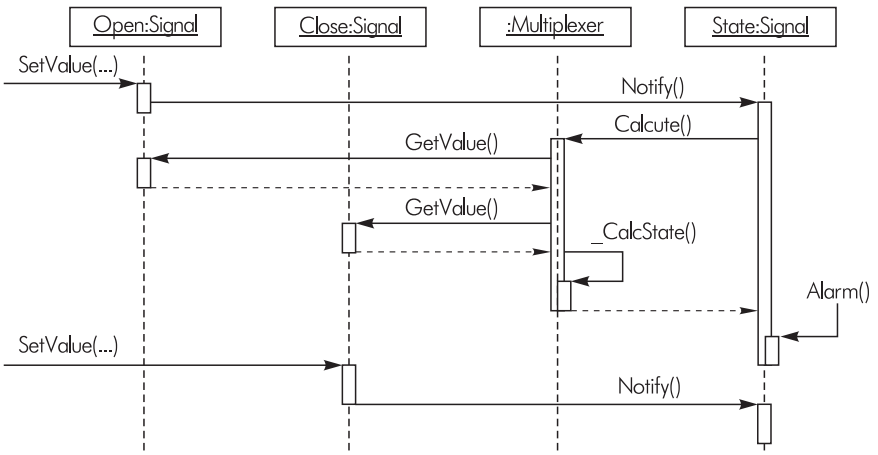

Рис. 3. Вариант алгоритма перерасчета агрегированного значения.

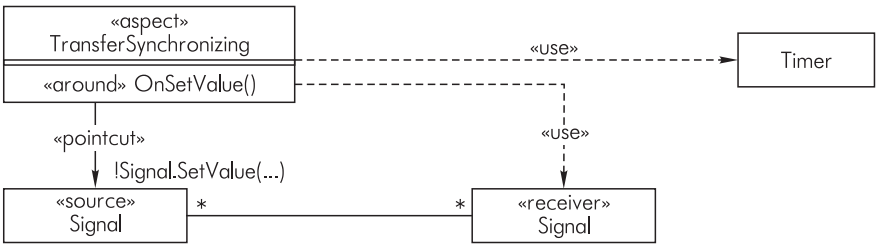

Рис. 4. Введение аспекта TransferSynchronizing для управления передачей данных.

Однако проблема остается, если источник данных – не один объект, а множество, и на диаграмме мы рассматриваем связь «многие-ко-многим». (Задача часто возникает при реализации субъектно-ориентированного подхода к проектированию БД ДП АСУТП, при котором вводится множество классов, содержащих наборы нормативно-справочных данных, описывающие один и тот же производственный объект при различных точках зрения (принципах декомпозиции предметной области), но которым требуется разделять оперативные данные о его состоянии). Передача (копирование) данных из источников получателям приводит к необходимости поддержания целостности.

Здесь мы приходим к классической ситуации введения аспекта – реализация в отдельном классе-источнике или получателе данных алгоритма отслеживания взаимо-

действия других пар объектов и синхронизации с ними исключительно затруднена. Цель – обеспечить синхронизацию обновления данных; желаемое поведение аспекта – реализовать «конвейерную передачу» обновлений по уровням иерархии и между поддеревьями БД.

Сначала определим «точку пересечения». Она одна – аспект перехватывает управление при попытке выполнения каким-либо из объектов-источников записи одного или нескольких новых значений; блокирует обработку новых значений в объектах-получателях, дает завершиться перехваченным методам SetValue(...), после чего ожидает в течение заданного интервала времени выполнения аналогичных действий другими объектами-источниками (см. рис. 4). По истечении времени ожидания происходит разблокирование всех объектов-получателей, в которые были записаны новые значения. Данная логика показана на рис. 5. Отношение «многие-ко-многим» классов-источников и получателей данных рассматривается как множество отношений «один-ко-многим» их экземпляров (рассматриваются множества объектов – экземпляров некоторых классов, объединенные в две группы по их роли в частном информационном взаимодействии: источников и получателей данных; стереотипы «source» и «receiver» являются производными от базового «group». Имена групп (при отличии стереотипа) совпадают, что подчеркивает единообразие входящих в них классов.

Отметим, что не требуется вводить точку пересечения с классом-получателем данных, поскольку нет необходимости перехватывать все выполняемые вызовы изменения данных в нем, а также использование стереотипа «around» для метода OnSetValue() аспектного класса: часть служебных операций выполняется до исполнения вызова SetValue(...), часть – после. Предложенное решение все еще содержит ряд недостатков: объект-

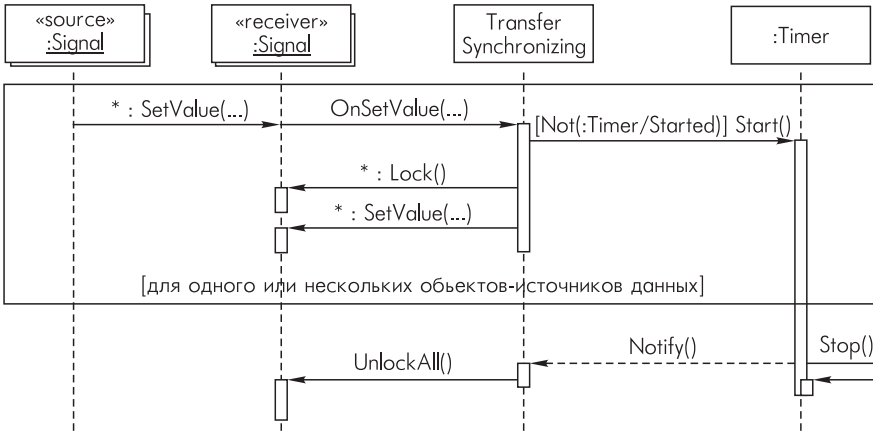

Рис. 5. Выполнение передачи данных при введенном аспекте TransferSynchronizing.

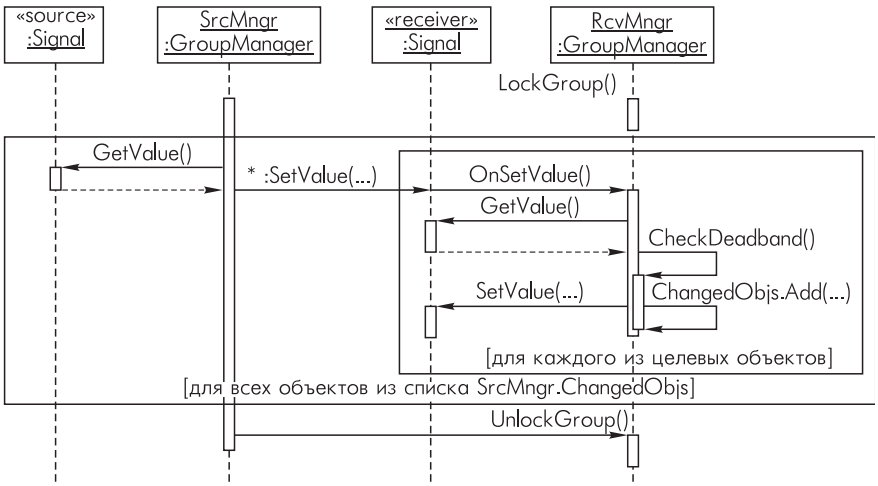

Рис. 6. Выполнение передачи данных при введенном аспекте GroupManager.

источник зависим от реализации объекта-получателя; введение периодов ожидания плохо согласуется с событийной моделью; объект-получатель должен проверять попадание разности нового и предыдущего значений в «зону нечувствительности»; его структура усложняется средствами поддержки блокировок. Можно предложить вариант, свободный и от данных недостатков.

Для этого определим, что у каждой группы есть менеджер – экземпляр аспектного класса, выполняющий контракты каждого из классов группы, в т.ч. и вызов метода SetValue(...) для записи измененного значения в класс-получатель. Это позволяет также выполнять операции блокирования/разблокирования только по отношению к этому аспектному классу. На него же может быть переложено выполнение проверки «существенности» отличия нового значения от текущего в объекте. Тогда желаемое поведение, реализующее транзакционный подход к передаче массива данных, можно отобразить в виде диаграммы (рис. 6), где показано взаимодействие только двух групп, хотя, разумеется, объекты и даже единственный объект группы-источника может записывать данные в несколько объектов, в т. ч. принадлежащих различным группам. Поскольку, как отмечалось выше, каждая из групп может играть роль как источника, так и получателя данных, то на диаграмме классов достаточно показать связь аспекта и одной группы (рис. 7а).

Диаграмма, показывающая внутреннюю логику аспекта при разблокировании, приведена на рис. 7б. Отметим, что возможна одновременная блокировка группы несколькими менеджерами групп-ис-

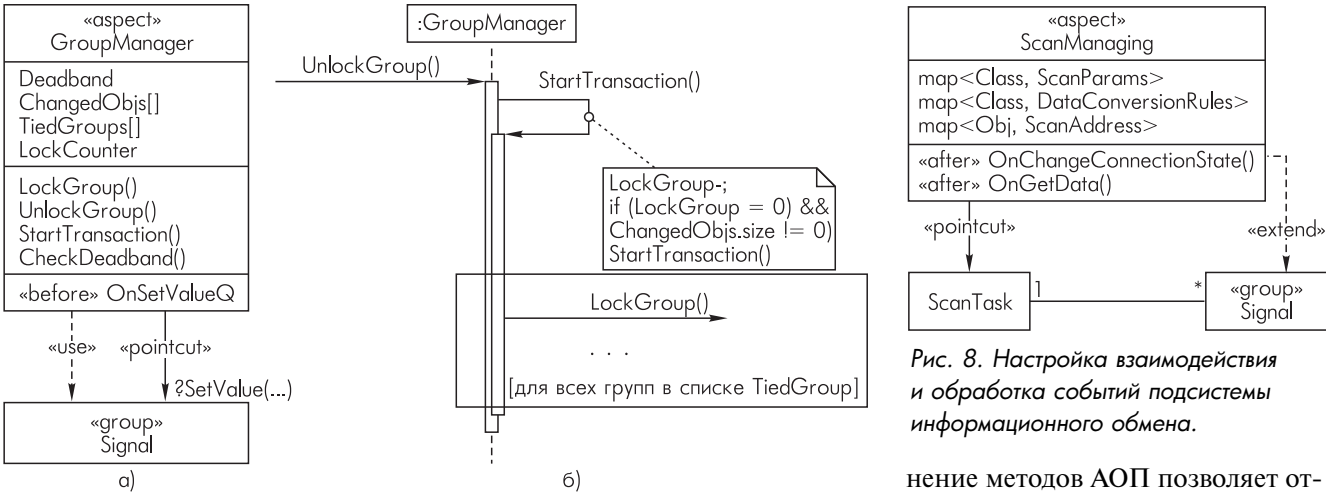

Рис. 7. Структура и внутренняя логика аспектного класса GroupManager.

Рис. 8. Настройка взаимодействия и обработка событий подсистемы информационного обмена.

точников: запись различных данных в один и тот же объект в любом случае не имеет смысла, а одновременная запись значений в различные объекты заблокированной группы несколькими источниками не приводит ни к каким отрицательным последствиям.

Взаимодействие с подсистемой информационного обмена

Взаимодействие «объект БД – модуль информационного обмена с внешними системами (далее – сканирования)» в ряде БД ДП АСУТП настраивается с использованием «списка сигналов»: в табличной форме объединяются индивидуальные параметры опроса каждого сигнала (тип данных, адрес в контроллере, периоды периодических опросов значений и изменений и т.п.). В других системах данная параметрическая информация сохраняется в самом объекте БД. Оба этих подхода обладают тем недостатком, что не используют естественно напрашивающееся предположение об одинаковости параметров (коммуникационных, преобразования «сырых» данных) для однотипных объектов БД, т.е. экземпляров одного класса. Поэтому, в данном случае аспектный класс используется как расширитель структуры класса БД (это отмечено отношением зависимости со стереотипом «extend»): он содержит таблицы соответствий класса и параметров сканирования, класса и правил преобразования полученных данных, а также список соответствий индивидуальных объектов и адресов во внешней системе (или правила получения адресов по имени объектов).

С другой стороны, от различных модулей сканирования требуется обеспечивать однотипное поведение при определенных событиях, например, качество значений всех объектов БД, получающих данные от внешней системы, с которой потеряна связь, должно быть установлено в «недостоверное». Аспектный класс «пересекает» операции модуля сканирования, результаты которых влияют на прочие объекты БД, и реализует единообразную логику для различных приложений информационного обмена.

Заключение

Иерархия классов отражает уровни абстракции сущностей предметной области, иерархия объектов (в БД ДП АСУТП) является информационной моделью автоматизируемого производства, ТП. Для отражения различных точек зрения возможно разделение классов и построение нескольких подмоделей, используя различные принципы декомпозиции. Применение методов АОП позволяет отделить средства реализации контрактов каждого из классов (механизмов поддержки в них различных ограничений и требований к информационной системе в целом) от описываемых ими абстракций сущностей, за счет чего повысить степень надежности БД, производительность обработки и передачи данных, возможность повторного использования проектных решений.

Список литературы

1. *Meyer, B.* Object-Oriented Software Construction. New-York, NY: Prentice Hall, 1988.
2. *Kiczales G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J.* Aspect-Oriented Programming. In proc. of ECOOP, 1997, LNCS 1241.
3. *Шукла Д., Селлз С.Ф.К.* АОП: Более эффективная инкапсуляция и повторное использование кода. // MSDN Magazine/Русская редакция. 2002. Спецвыпуск №1.
4. *Stein, D., Hanenberg, S., Unland, R.* Designing Aspect-Oriented Crosscutting in UML. In proc. of Workshop on Aspect-Oriented Modeling with UML at AOSD, 2002.
5. *Pawlak, R., Duchien, L., Florin, G., Legond-Aubry, F., Seinturier, L., Martelli, L.* A UML Notation for Aspect-Oriented Software Design. In proc. of Workshop on Aspect-Oriented Modeling with UML at AOSD, 2002.
6. *Богданов Н.К.* Тиражируемые программные комплексы для создания АСУТП. // Промышленные АСУ и контроллеры. 2000, №12.

Богданов Николай Константинович *– инженер компании АтлантикТрансГазСистема.*
bogdanov@atgs.ru