МАТЕМАТИЧЕСКИЕ ОСНОВЫ КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ

DOI 10.17223/20710410/13/4

УДК 004.4'2+004.43

ВНЕДРЕНИЕ ПОЛИТИК БЕЗОПАСНОСТИ В ПРОГРАММНЫЕ СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ

Д. А. Стефанцов

Томский государственный университет, г. Томск, Россия

E-mail: d.a.stefantsov@isc.tsu.ru

Рассматривается проблема защиты систем обработки информации (СОИ) посредством интеграции их с политиками безопасности (ПБ). Проанализированы существующие методы решения этой проблемы, отмечены их недостатки и предложен оригинальный метод её решения с помощью аспектно-ориентированного программирования (АОП), лишённый этих недостатков. В отличие от традиционных реализаций АОП, в данном методе аспект ПБ присоединяется к СОИ посредством соединительного модуля без изменения программных модулей СОИ и ПБ, написанных независимо друг от друга и от соединительного модуля. Для реализации этого метода созданы инструментальные средства в составе языка АОП AspectTalk, виртуальной машины и транслятора с языка AspectTalk в язык виртуальной машины. Работа содержит краткое описание предложенного метода и перечисленных инструметальных средств его реализации.

Ключевые слова: *система обработки информации, политика безопасности, аспектно-ориентированное программирование, AspectTalk, виртуальная машина.*

Введение

Защита информации, хранимой и преобразуемой в системах обработки информации (СОИ), является актуальной проблемой с момента появления многопользовательских компьютерных систем [1]. При разработке защиты таких систем определяется модель нарушителя в виде формального описания набора угроз системе и/или атак на неё, а также политика безопасности (ПБ) в виде набора формальных правил противодействия последним [2]. Невозможность компрометации системы при условии следования правилам ПБ доказывается соответствующими теоремами безопасности [1 – 3].

Одной из первых формальных ПБ, разработанных для реализации в вычислительных системах, является модель Белла — ЛаПадулы [1]. Обзор большинства существующих моделей безопасности можно прочитать в [3]. Министерством обороны США представлена классификация вычислительных систем на основе реализации ПБ для определённых моделей нарушителя [4].

Помимо политик разграничения доступа, в понятие ПБ будем включать любые требования к защите СОИ, например использование криптографических средств защиты информации и специальных методов преобразования информации, таких, как фильтрация и кодирование.

Программную составляющую защищённой СОИ можно разделить на две части (подсистемы) — часть, реализующую целевую обработку информации (далее — часть

ОИ), и часть, реализующую программную модель ПБ (далее — часть ПБ). Примерами подобной модели могут служить следующие алгоритмы и структуры данных, реализующие действия, предписываемые ПБ:

1) учётные записи пользователей, списки прав доступа — при реализации политики разграничения доступа;
2) алгоритмы шифрования и цифровой подписи, криптографические протоколы, хранилища ключевой информации — при реализации криптографической защиты данных;
3) алгоритмы фильтрации вводимых данных для предотвращения SQL-инъекций и XSS-атак.

Для соединения этих частей в одну программу часть ОИ обычно изменяется таким образом, что при совершении действий по обработке информации производится обращение к части ПБ для определения возможности доступа к запрашиваемой информации или для её преобразования. Эти изменения делают текст программы части ОИ зависимым от текста программы части ПБ.

При изменении модели нарушителя соответствующие изменения вносятся в часть ПБ, что может повлечь за собой необходимость изменения части ОИ. Примером подобных изменений может служить реализация политики мандатного разграничения доступа SELinux для операционной системы (ОС) GNU/Linux, ранее обладавшей только дискреционной политикой разграничения доступа [5]. Тесная интеграция программных реализаций СОИ и ПБ является препятствием к внесению изменений в ПБ.

Примером СОИ, в которой части ОИ и ПБ реализованы раздельно, может служить ОС Mac OS X 10.4. В ней при необходимости организации доступа субъекта системы к её объекту вызывается специальная функция подсистемы `kauth` [6]. Эта функция опрашивает множество специальных модулей, ответственных за реализацию политики разграничения доступа. На основании ответов, полученных от модулей, функция вычисляет ответ подсистемы безопасности на запрос доступа: разрешение или отказ. Модули, выносящие решение в соответствии с некоторой ПБ, разрабатываются и реализуются независимо от ядра ОС в соответствии со специальными правилами и могут быть загружены в память ОС администратором системы. Таким образом, для реализации некоторой политики разграничения доступа с помощью подсистемы `kauth` необходимо описать совокупность модулей на некотором языке программирования, скомпилировать их и загрузить в память ОС. На рис. 1 схематически изображена работа ОС с реализованной в ней подсистемой `kauth`.

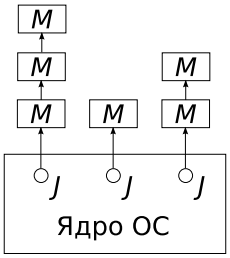

Рис. 1. Подсистема `kauth` ОС Mac OS X 10.4: цепочки модулей M определяют возможность предоставления доступов J

Подсистема `kauth` реализована также в ОС NetBSD [7]. В работе [8] показана возможность реализации механизма `jail` ОС FreeBSD в ОС NetBSD средствами подсистемы `kauth`.

К недостаткам подсистемы `kauth` можно отнести невозможность реализации ПБ, не являющейся политикой разграничения доступа.

Автором разработана технология и инструментальная среда создания защищённых СОИ в виде совокупности независимых частей ОИ и ПБ произвольного вида, соединяемых способом, исключающим необходимость их изменения. В основе этого способа лежит аспектно-ориентированное программирование (АОП) [9], модифицированное для случая, когда одна из объединяемых подсистем реализует ПБ, а именно: интеграция частей ОИ и ПБ осуществляется при помощи простых соединительных модулей, которые зависят одновременно от текста программы ОИ и текста присоединяемого аспекта ПБ. Части ОИ и ПБ и соединительные модули представляются в виде исходных текстов на языке АОП AspectTalk [10], специально разработанном для этой цели. В нём исходные тексты частей ОИ и ПБ являются описаниями классов объектов их предметных областей, а соединительные модули устанавливают соответствие между классами этих частей. В процессе трансляции соответствующие классы объединяются в один. Трансляция выполняется в язык интерпретируемой виртуальной машины (ВМ).

Краткое сообщение об этих средствах создания защищённых СОИ можно найти в [11]. Более развёрнутое изложение основных элементов данной технологии является целью настоящей работы.

1. Краткая характеристика АОП

АОП — это способ программирования, при котором главная подсистема (часть, выполняющая основную функцию системы) может быть реализована независимо от подчинённой подсистемы (части, выполняющей дополнительную функцию — например, реализацию ПБ) [10].

Рассмотрим СОИ, реализованную с помощью традиционного процедурного подхода и состоящую из программы P и библиотеки L (рис. 2). Если в точках J программы P необходим вызов процедуры F библиотеки L, он будет осуществлён явно, что сделает текст программы P зависимым от текста библиотеки L. В случае замены или удаления библиотеки L необходимо изменение текста программы P.

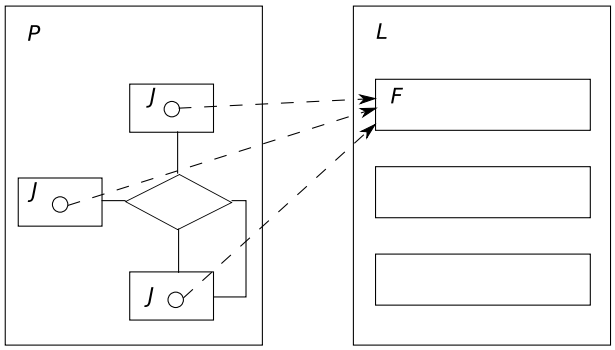

Рис. 2. Использование традиционного подхода в программировании. Текст программы P зависит от текста библиотеки L

Разработка программ с помощью АОП основана на использовании неявных вызовов процедур. Рассмотрим СОИ, состоящую из программы P и аспекта A (рис. 3). Аспект — это библиотека специального вида, состоящая из описаний состояний программы, а также специальных алгоритмов. Между описаниями и алгоритмами устанавливается соответствие: всякий раз, когда программа достигает состояния, подхо-

дящего под описание C, запускается соответствующий этому описанию алгоритм D. При этом J — это точки выполнения программы, в которых достигаются состояния, подходящие под описание C, и осуществляется неявный вызов алгоритма D. В данном случае текст аспекта A зависит от текста программы P, к которой этот аспект применяется, но текст программы P не зависит от текста аспекта A. К недостаткам данного варианта АОП можно отнести зависимость аспектов от программы, к которой они применяются, что затрудняет перенос аспектов в другие СОИ [12].

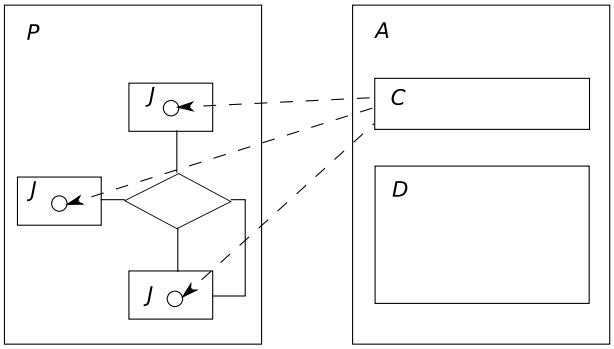

Рис. 3. Использование АОП. Текст аспекта A зависит от текста программы P

АОП не является самостоятельным способом программирования, но реализуется в виде расширения некоторого традиционного способа программирования [12]. А именно, программа P и алгоритм D могут быть реализованы без применения аспектов. Аспектное расширение традиционного подхода — это присутствие описаний C состояний программы P.

2. Метод защиты СОИ с помощью АОП

Опишем метод создания защищённых СОИ средствами АОП, в котором тексты программы и аспектов разрабатываются независимо друг от друга, а для интеграции частей используются специальные соединительные модули, которые зависят одновременно от текста программы и текста присоединяемого аспекта (рис. 4). Соединительные модули предполагаются много проще соединяемых ими частей СОИ.

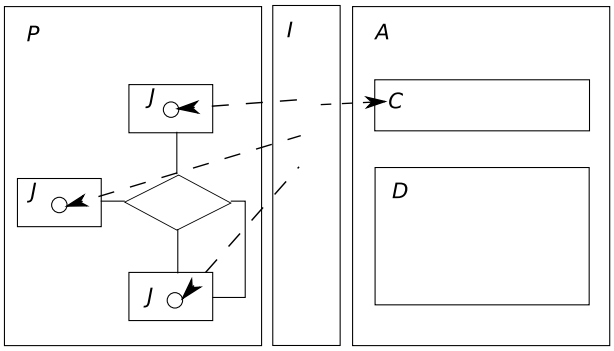

Рис. 4. СОИ получена интеграцией частей с помощью соединительных модулей

Части ОИ и ПБ предполагаются представленными в виде исходных текстов на объектно-ориентированном языке программирования. Предметной областью части ОИ является целевая предметная область СОИ, предметной областью части ПБ является политика безопасности. Исходные тексты частей ОИ и ПБ — это описания классов объектов соответствующих предметных областей.

Предлагаемая технология основана на установлении соответствия между классами объектов данных предметных областей. На рис. 5 показано соответствие между классами двух предметных областей: Unix-подобной ОС и дискреционной политики разграничения доступа.

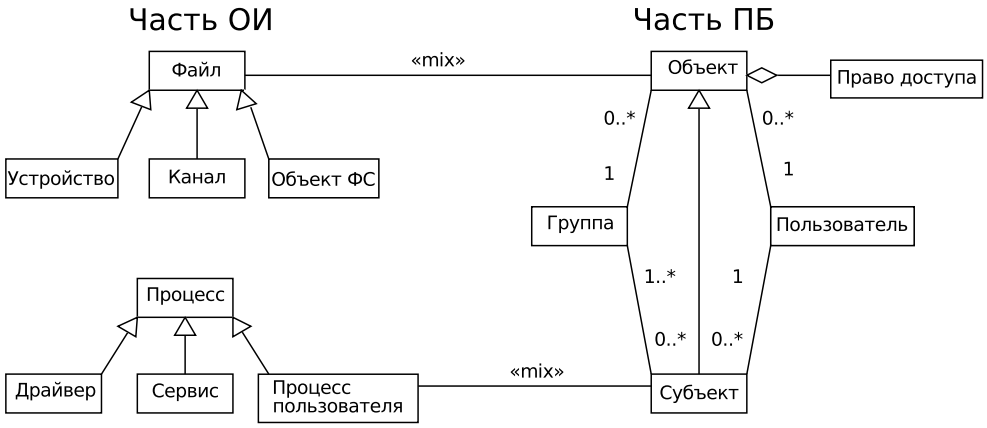

Рис. 5. Диаграмма классов и примесей Unix-подобной системы обработки файлов

Классы подчинённой системы (в данном случае — ПБ) называются примесями, а соответствие примесей части ПБ и классов части ОИ обозначается «mix». По данному соответствию на этапе трансляции происходит объединение классов и примесей в составной класс, множество член-данных которого — это объединение множеств член-данных соответствующих класса и примесей, а множество член-функций — объединение множеств член-функций соответствующих класса и примесей (рис. 6). Соответствие «mix» является альтернативной реализацией механизма множественного наследования.

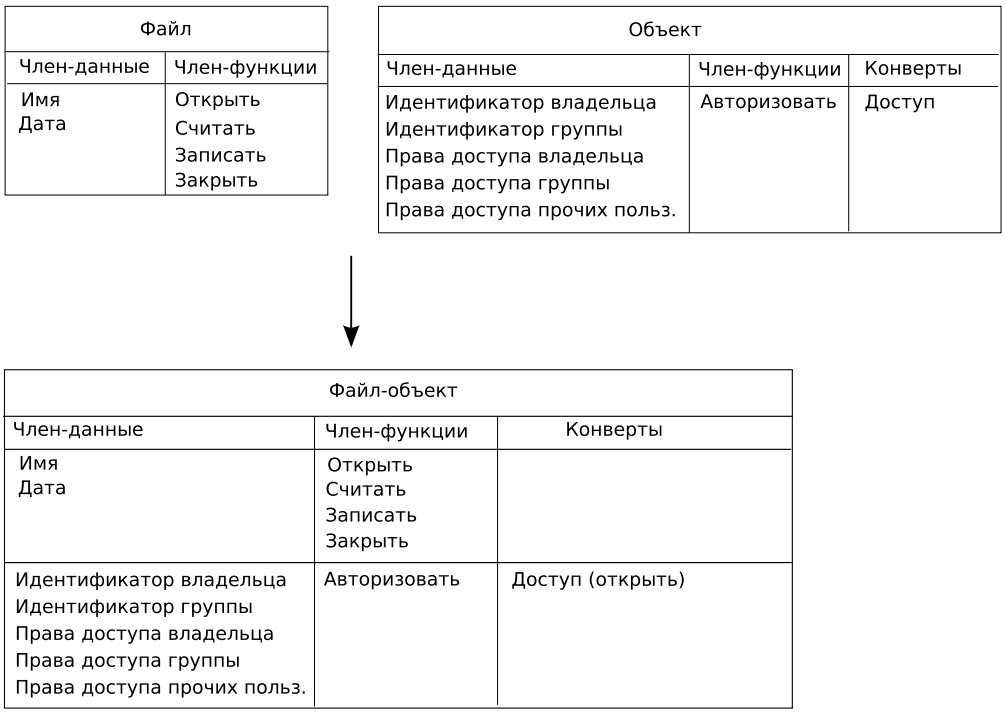

Рис. 6. Объединение класса и примеси на этапе трансляции

Для примесей определяются также специальные алгоритмы, называемые конвертами. При указании соответствия между классами и примесями для конвертов задаются шаблоны строк в виде регулярных выражений. Всякий раз, когда будет вызвана член-функция с именем α некоторого объекта из части ОИ, α будет проверено на соответствие шаблонам для конвертов соответствующих примесей. При совпадении вместо вызываемого алгоритма будет запущен соответствующий алгоритм-конверт. Алгоритм-конверт получает в качестве параметров объект, вызвавший исходную член-функцию, объект, член-функция которого была вызвана, имя член-функции, а также параметры вызова. Алгоритм-конверт формирует возвращаемое значение, в процессе чего может вызывать исходную член-функцию. Механизм конвертов позволяет реализовать вход программы в аспект.

Рассмотренный механизм примесей похож на описанный в [12], однако в [12] примесь реализована в виде базовой конструкции языка. Предлагаемый же подход основан на реализации в языке протокола метаобъектов [13] в качестве базовой конструкции с последующей реализацией примесей на самом языке программирования. Это позволяет упростить семантическую модель языка и не различать примеси, модифицирующие алгоритмы, и примеси, модифицирующие структуры данных, как это сделано в [12].

Таким образом, реализован следующий метод построения СОИ, защищённых ПБ:

1) часть ОИ описывается в терминах классов;
2) часть ПБ описывается в терминах примесей;
3) соединительные модули описываются в виде сопоставлений класс — примесь, а также указания шаблонов для алгоритмов-конвертов.

3. Краткая характеристика языка AspectTalk

Для создания СОИ, защищённых с помощью АОП, необходим специальный аспектно-ориентированный язык программирования (АОЯП), на котором описываются программа и её аспекты. Первым АОЯП принято считать AspectJ [14, 15] — расширение объектно-ориентированного языка программирования Java. В AspectJ описания C состояний программы P (см. рис. 3) даются в виде набора синтаксических конструкций, и состояние считается достигнутым, если выполняется действие, соответствующее той или иной конструкции языка. В других АОЯП, например в MetaclassTalk [12], под состоянием понимается состояние специальных переменных. О других способах реализации АОП можно прочитать в [16 – 18].

Для экспериментального исследования изложенного выше метода создания защищённых СОИ разработан и реализован АОЯП AspectTalk [10]. За его основу взят диалект Little Smalltalk объектно-ориентированного языка программирования Smalltalk [19].

Все типы данных в AspectTalk являются объектами. Среди объектов выделяется подмножество, называемое классами. С помощью классов и иерархии их наследования реализуются механизмы создания объектов и установления соответствия между объектами и их член-данными и член-функциями.

В языке AspectTalk декларативные конструкции языка Smalltalk, отвечающие, в том числе, за объявление классов и анализируемые на этапе трансляции, заменены на совокупности элементарных операций, которые исполняются на этапе выполнения программы. Это позволяет уменьшить число конструкций языка.

Язык AspectTalk содержит следующие базовые операции:

1) примитивы ВМ, с помощью которых производятся низкоуровневые действия, такие, как сложение чисел, работа с файлами;
2) посылка сообщения — это базовая операция, которая приводит к вызову член-функции, определяемой в соответствии с иерархией наследования классов;
3) возврат результата — операция, приводящая к завершению работы член-функции и возврату выполнения программы на следующую команду после соответствующей посылки сообщения;
4) присвоение — операция, с помощью которой переменным присваиваются указатели на объекты.

Как и в Smalltalk, в AspectTalk есть специальный тип данных — метаклассы, но, в отличие от Smalltalk, метаклассы в AspectTalk не только являются классами классов, но и позволяют программисту давать ВМ дополнительные указания о работе объектной системы. Эти указания определяются в виде обработчиков операции посылки сообщения от объекта к объекту и наследуются метаклассами в иерархии наследования метаклассов. Более подробное описание языка AspectTalk с примером реализации с его помощью некоторой ПБ можно найти в [10].

4. Краткая характеристика интерпретатора

В большинстве случаев АОЯП разделяется на две составляющие: базовый язык программирования, с помощью которого реализуются основная часть программы и алгоритмы аспектов, и аспектное расширение, с помощью которого аспекты присоединяются к программе. Транслятор с такого языка реализуется, в основном, следующим образом. Этап трансляции разделяется на два шага. На первом шаге программа, написанная на базовом языке программирования совместно с его аспектным расширением, транслируется в программу, содержащую только конструкции базового языка программирования. В результате выполнения этого шага все неявные вызовы процедур аспекта, которые могут быть определены без запуска программы, заменяются на явные в автоматическом режиме. Если вызов процедуры может быть осуществлён на основе информации, доступной только во время выполнения программы, то во всех предположительных местах вызова помещается обращение к специальной библиотеке, которая определяет необходимость вызова процедуры аспекта. На втором шаге осуществляется трансляция с базового языка программирования на язык машины, выполняющей программу.

Возможен другой способ реализации транслятора с АОЯП, используемый в данной работе, — в виде ВМ, внутренние структуры данных и алгоритмы которой реализуют программную модель АОЯП, и транслятора с АОЯП в язык ВМ. В этом случае трансляция осуществляется за один шаг — программа, написанная на базовом языке программирования и его аспектном расширении, транслируется в язык ВМ напрямую. Совокупность транслятора и ВМ далее будем называть интерпретатором. Интерпретация менее эффективна по времени и по памяти, чем компиляция, однако реализуется более простым способом и более наглядна.

В интерпретаторе с языка AspectTalk во многом повторяется структура интерпретатора с языка Smalltalk [19]. Основные принципы, применяемые для перевода программы с AspectTalk в язык ВМ, не новы и описаны в [20].

ВМ интерпретатора с языка AspectTalk является стековой машиной, язык которой представляет собой польскую инверсную запись команд, с помощью которых производится манипуляция данными, а также обращение к ОС, в которой запускается интерпретатор. Основной тип данных ВМ — объект. Экземпляры этого типа данных

помещаются и извлекаются с вершины стека, а также преобразуются специальными командами.

Команды ВМ можно разделить на следующие группы:

— примитивы ВМ;
— команды создания объектов;
— команды, помещающие значение переменной на вершину стека;
— команды, сохраняющие в переменных объект с вершины стека;
— команды, посылающие сообщение объекту, находящемуся на вершине стека (при этом параметры сообщения также берутся из стека);
— команда возврата из процедуры;
— команды удаления и дублирования объекта на вершине стека.

Примитивы ВМ — это операции обращения к ВМ для выполнения низкоуровневых операций, такие, как сложение целых чисел, выделение памяти, вывод строки на экран, работа с файлами и т. д. Поиск переменных по имени осуществляется в соответствии с моделью лексического окружения, описанного в [21].

Основа работы интерпретатора — объектная модель. В этой модели вся система представлена в виде совокупности объектов, посылающих сообщения другим объектам. Получив сообщение, объект осуществляет его диспетчеризацию — поиск процедуры, которая должна быть выполнена в ответ на сообщение, после чего объект возвращает результат работы процедуры отправителю сообщения. Совокупность сообщений, для которых объект может найти соответствующую процедуру, называется протоколом этого объекта. В большинстве систем существует множество объектов, обладающих одинаковым протоколом. Для того чтобы упростить алгоритм поиска процедур для таких объектов, в объектно-ориентированном программировании вводится специальное понятие — класс. Класс — это объект, который осуществляет диспетчеризацию сообщений для объектов, обладающих одинаковым протоколом. Такие объекты называются экземплярами этого класса. Вводится также отношение наследования, которое организует классы в иерархию: потомок может обработать те же сообщения, что и предок, и ещё некоторые дополнительные сообщения.

В свою очередь, метаклассы — это классы классов. Они не только осуществляют диспетчеризацию сообщений, отправляемых классам, но и могут заменить у некоторых классов процедуру диспетчеризации сообщений, посылаемых экземплярам этих классов. Этот механизм называется протоколом метаобъектов и описан в [13]. С помощью этого механизма реализовано АОП в AspectTalk: всякий раз, когда некоторому объекту посылается сообщение, метакласс прерывает его диспетчеризацию и проверяет посылаемое сообщение на соответствие набору регулярных выражений. Если соответствие найдено, то запускается специальный алгоритм-конверт, соответствующий регулярному выражению. Если соответствия не найдено, то метакласс возобновляет диспетчеризацию сообщений через иерархию классов.

Следует отметить, что диспетчеризация сообщений, классы, метаклассы, протоколы метаобъектов и конверты не являются встроенными конструкциями языка ВМ, а описаны на самом AspectTalk в его библиотеке: для этого достаточно четырёх базовых операций языка.

Заключение

Разработаны технология и инструментальная среда создания защищённых СОИ в виде совокупности независимых частей (подсистем) — ОИ и ПБ произвольного вида,

соединённых простым способом, основанном на АОП, модифицированном для случая, когда одна из подсистем — это реализация ПБ. В составе разработанных средств:

1) язык программирования AspectTalk, обладающий конструкциями базового языка программирования Smalltalk и метаязыковыми конструкциями;
2) ВМ, выполняющая операции над данными, описанными на языке AspectTalk;
3) транслятор с языка AspectTalk в язык ВМ.

Разработанная технология предполагает:

1) описане ОИ в терминах классов;
2) описание ПБ в терминах примесей;
3) описание соединительных модулей путём сопоставлений класс — примесь и указания шаблонов для алгоритмов-конвертов.

Научная новизна исследования состоит в модификации аспектно-ориентированного подхода в программировании, состоящей в специализации средств метаязыка, а именно: в классическом АОП они используются при написании аспекта совместно с операциями объединения с основной программой, в модификации — только при написании соединительных модулей. Последнее упрощает процедуру повторного использования аспектов: вместо переписывания заново самого аспекта (в классическом подходе) переписывается только соединительный модуль (в новом подходе), который, как правило, много проще присоединяемого аспекта.

Использование разработанных методов и средств позволяет снизить затраты на внесение изменений в ПБ СОИ, а также повторно использовать реализации ПБ при разработке новых программных систем.

ЛИТЕРАТУРА

1. *Bell D. E. and LaPadula L. J.* Secure computer system: Unified exposition and multics interpretation: Tech. Rep. ESD-TR-75-306. The MITRE Corporation, 1976.
2. *Landwehr C. E.* Formal models for computer security // ACM Comput. Surv. 1981. V. 13. No. 3. P. 247–278.
3. *Девянин П. Н.* Анализ безопасности управления доступом и информационными потоками в компьютерных системах. М.: Радио и связь, 2006. 176 с.
4. DoD 5200.28-STD (Trusted Computer System Evaluation Criteria) USA: National Computer Security Center, 1985. 116 p.
5. `http://www.nsa.gov/research/selinux/index.shtml` — Security-Enhanced Linux. 2009.
6. `http://developer.apple.com/library/mac/#technotes/tn2127/_index.html` — Technical Note TN2127. Kernel Authorization. 2010.
7. `http://netbsd.gw.com/cgi-bin/man-cgi?kauth+9+NetBSD-current` — NetBSD Kernel Developer's Manual. kauth. 2009.
8. `http://2008.asiabsdcon.org/papers/P3A-paper.pdf` — Implementing Jails Under the kauth Framework. 2008.
9. *Elrad T., Filman R. E., and Bader A.* Aspect-Oriented Programming // Commun. ACM. 2001. V. 44. No. 10. P. 29–32.
10. *Стефанцов Д. А.* Реализация политик безопасности в компьютерных системах с помощью аспектно-ориентированного программирования // Прикладная дискретная математика. 2008. № 1. С. 94–100.
11. *Стефанцов Д. А.* Технология и инструментальная среда создания защищённых систем обработки информации // Прикладная дискретная математика. Приложение. 2009. № 1. С. 55–56.

12. *Bouraqadi N., Seriai A., and Leblanc G.* Towards unified aspect-oriented programming // ESUG 2005 Research Conference. Brussels, Belgium, 2005. 22 p.
13. *Kiczales G.* The Art of Meta-Object Protocol. The MIT Press, 1991. 345 p.
14. `http://eclipse.org/aspectj` — The AspectJ Project. 2011.
15. *Kiczales G., Hilsdale E., Hugunin J., et al.* Getting Started with AspectJ // Commun. ACM. 2001. V. 44. No. 10. P. 59–65.
16. *Diaz Pace J. A. and Campo M. R.* Analyzing the Role of Aspects in Software Design // Commun. ACM. 2001. V. 44. No. 10. P. 67–73.
17. *Lieberherr K., Orleans D., and Ovlinger J.* Aspect-Oriented Programming with Adaptive Methods // Commun. ACM. 2001. V. 44. No. 10. P. 39–41.
18. *Bergmans L. and Aksit M.* Composing Crosscutting Concerns Using Composition Filters // Commun. ACM. 2001. V. 44. No. 10. P. 51–57.
19. *Goldberg A. and Robson D.* Smalltalk 80 — The Language and its implementation. Addison-Wesley, 1983. V. 1. 714 p.
20. *Ахо А., Ульман Дж., Сети Р.* Компиляторы: принципы, технологии и инструменты. М.: Вильямс, 2003. 768 с.
21. *Абельсон Х., Сассман Дж.* Структура и интерпретация компьютерных программ. М.: Добросвет, 2006. 608 с.