2008 Математические основы компьютерной безопасности № 1(1)
DOI 10.17223/20710410/1/14
УДК 681.322

РЕАЛИЗАЦИЯ ПОЛИТИК БЕЗОПАСНОСТИ В КОМПЬЮТЕРНЫХ СИСТЕМАХ С ПОМОЩЬЮ АСПЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Д.А. Стефанцов

Томский государственный университет

E-mail: d.a.stephantsov@gmail.com

Рассматривается аспектно-ориентированное программирование как средство реализации политик безопасности в компьютерных системах. Даётся краткий обзор аспектно-ориентированного языка программирования AspectJ с примером реализации аспектов безопасности с помощью этого языка. Вводятся понятия из областей объектно-ориентированного программирования и метаобъектных протоколов. Показывается, как на основе метаобъектного протокола может быть построено аспектно-ориентированное программирование. Даётся краткий обзор аспектно-ориентированного языка AspectTalk, разработанного автором статьи, а также рассматривается пример реализации простой политики безопасности с помощью этого языка.

Ключевые слова: *политика безопасности*, *аспектно-ориентированное программирование*, *объектно-ориентированное программирование*, *метаобъектный протокол*, *AspectJ*, *AspectTalk*, *MetaclassTalk*, *Smalltalk.*

В связи с проникновением компьютерной техники в производство, образование и повседневную жизнь резко возрастает потребность в защите вычислений от случайных или умышленных вмешательств. В данной статье будем рассматривать только умышленные вмешательства. Поэтому *угрозу* можно определить как цель злоумышленника, а *атаку* – как способ достижения этой цели.

Безопасностью называется состояние защищённости системы от некоторых видов угроз и/или атак. Это определение подразумевает, что безопасность – относительное понятие, т.е. та или иная защита – это защита от конкретных атак (и, возможно, от некоторых угроз) [1].

Как правило, для защиты компьютерных систем поступают следующим образом: вводятся формальные определения компьютерной системы и некоторой угрозы. Затем предъявляется ряд требований, соблюдение которых влечёт невозможность реализации данной угрозы в данной системе. Такие требования формулируются в виде *теорем безопасности* [2]. Наличие формальных требований к системе в большинстве случаев даёт возможность применения в реальных компьютерных системах алгоритмов, приводящих к соблюдению этих требований.

Политикой безопасности будем называть ряд требований, предъявляемых к компьютерной системе с тем, чтобы привести её в состояние безопасности. *Реализацией политики безопасности* будем называть реализацию на каком-либо языке программирования алгоритмов, обуславливающих соблюдение требований политики безопасности.

Реализации политик безопасности можно встретить во многих компьютерных системах – от операционных систем общего назначения [3] до узкоспециализированных Web-систем хранения данных. Несмотря на годы практики, большинство методов реализации политик безопасности являются грубым нарушением принципа «разделения аспектов» изучаемого предмета [4].

Некоторые исследователи способов проектирования программного обеспечения (ПО) сходятся во мнении, что данное нарушение – следствие ограничений, накладываемых современными средствами реализации, в частности существующими парадигмами программирования [5, 6]. В этих же источниках указывается один из возможных путей решения данной проблемы – аспектно-ориентированное программирование (АОП). Данная статья посвящена введению в это средство разработки ПО и реализации с его помощью политик безопасности компьютерных систем.

Процесс реализации защищённого приложения традиционными методами предполагает как совместное проектирование политик безопасности и основной части приложения, так и их совместную реализацию. Если впоследствии появится необходимость в изменении политики безопасности, то внедрение этих изменений будет осложнено необходимостью анализа, повторного проектирования и реализации всех точек пересечения (модулей) аспекта безопасности и главной части приложения. Отметим, что изменение политики безопасности зачастую является единственной адекватной мерой противодействия новым видам атак и/или угроз, и, несмотря на это, политики безопасности современных компьютерных систем остаются неизменными. Вероятно, причиной тому служит излишняя сложность внесения изменений.

АОП является расширением существующих парадигм программирования: аспектно-ориентированный язык обладает как конструкциями, характерными для традиционных языков, так и конструкциями, с помо-

щью которых описываются аспекты приложения. При использовании АОП основная часть приложения и его аспекты проектируются и реализуются независимо друг от друга и объединяются в одно целое в автоматическом режиме. Разработка основной части осуществляется, как правило, объектно-ориентированными конструкциями языка программирования, а аспектов – дополнительными, аспектно-ориентированными, конструкциями языка. Объединение этих частей в приложение производится либо виртуальной машиной, либо механизмами текстовых подстановок. В случае необходимости внесения изменений в политику безопасности они отразятся только на исходном коде аспектов приложения, но не на основной его части. Поскольку аспекты в АОП реализуются изолированно, эти действия производятся относительно простым для разработчика образом.

1. Аспектно-ориентированное программирование

По мнению авторов АОП, основным принципом разработки новых парадигм программирования служил принцип эффективной декомпозиции приложения на модули [7]. Однако ни одна из существующих парадигм не предложила способа разделения реализации различных аспектов одного приложения.

Понятие «аспект» понимается так же, как в [4], а именно как точка зрения на изучаемый предмет, рассматриваемая в изоляции от других точек зрения. Так, один из аспектов приложения (помимо основного назначения) – это безопасность (защищённость от конкретных атак).

Если, в силу декомпозиции системы, один и тот же модуль должен реализовывать алгоритмы двух или более аспектов, то говорят о пересекающихся аспектах. АОП – это парадигма программирования, целью которой является введение уровня модульности, позволяющего реализовывать различные аспекты системы изолированно друг от друга, независимо от того, пересекаются ли они или нет при первоначальной декомпозиции [5].

Реализация ПО методом АОП заключается в раздельном описании каждого аспекта системы (в том числе политики безопасности) и указании способа соединения различных аспектов в единое приложение [7]. Соединение различных аспектов в приложение осуществляется автоматически, средствами аспектно-ориентированных средств разработки.

АОП не является заменой какой-либо другой парадигмы программирования, но её расширением [5]. Типичный процесс создания приложения с помощью аспектно-ориентированных технологий выглядит следующим образом: производится декомпозиция системы на модули привычными (не аспектно-ориентированными) методами, выделяются модули, по которым пересекаются несколько аспектов приложения, один аспект объявляется главным (он выполняет основное назначение системы; такой аспект в будущем будем называть главным) – он реализуется привычными средствами (без АОП), прочие аспекты реализуются с помощью АОП.

Авторами АОП был взят за основу объектно-ориентированный язык программирования Java. Соответствующая аспектно-ориентированная версия языка носит название AspectJ [8].

Для иллюстрации применения АОП рассмотрим следующий фрагмент программы на языке Java (листинг 1).

```
public class Process {
    ...
    public void readAndPrint(File f) {
        if (this.id == f.id)
            System.out.println(f.getInfo());
        else
            System.out.println("This is not my file!");
    }
    ...
}
```

Листинг 1. Фрагмент программы на языке Java

Как видно, этот модуль (метод readAndPrint объекта класса Process) реализует алгоритмы сразу двух аспектов – главного аспекта (вывод на экран содержимого объекта класса File) и действий, связанных с обеспечением безопасности (проверка совпадения идентификаторов).

Требования, предъявляемые двумя аспектами к одному модулю, могут быть изолированы друг от друга средствами аспектно-ориентированного языка AspectJ, являющегося расширением языка Java (см. листинг 2).

В данном случае реализация чтения из файла изолирована от проверки совпадения идентификаторов. Служебное слово aspect объявляет новый аспект Security. Конструкция around указывает, что вместо метода void Process.readAndPrint (File) должен быть выполнен алгоритм, указанный далее в фигурных скобках. Алгоритм состоит в проверке совпадения идентификаторов. В случае успеха с помощью служебного слова proceed производится исполнение метода readAndPrint объекта p класса Process с аргументом f класса File.

```
public class Process {
    ...
    public void readAndPrint(File f) {
        System.out.println(f.getInfo());
    }
    ...
}
...
public aspect Security {
    ...
    void around(Process p, File f):
        call(void Process.readAndPrint(File)) && args(f) && target(p) {
            if (f.id == p.id)
                proceed(p, f);
            else
                System.out.println("That's not my file!");
    }
    ...
}
```

Листинг 2. Фрагмент программы на языке AspectJ

Следует отметить, что АОП не обязательно реализуется лингвистически, то есть в виде языка программирования. В [9] описываются другие подходы, в частности создание АОП-подобной архитектуры приложения. О различных подходах к реализации АОП можно прочитать также в [10–12].

2. Реализация аспектно-ориентированных средств разработки

В настоящее время возраст аспектно-ориентированных технологий превышает десять лет [13]. За это время образовалось сообщество исследователей, изучающих различные вопросы аспектно-ориентированного программирования [14]. Исследования поддерживаются крупными организациями, такими, как ACM.

Опишем один из подходов, применявшийся в [15, 16]. Первоначально аспектно-ориентированное программирование являлось расширением объектно-ориентированного программирования (ООП). Дадим определение *объекта* в объектно-ориентированной вычислительной системе.

Пусть K – непустое множество, которое будем называть множеством селекторов. Объект O есть тройка (S, M, s), где S – непустое множество, называемое множеством состояний объекта, s O S – начальное состояние объекта, M – множество пар (k, m), где k O K, m – алгоритм, представляющий собой последовательность команд для некоторой машины (которую будем называть *виртуальной машиной*) одного из следующих четырёх типов. Будем обозначать $s(t)$ O S состояние объекта O в момент времени t.

1. Команда изменения состояния. В результате выполнения команды состояние объекта изменится с $s(t)$ на $s(t + 1)$.

2. Команда посылки сообщения. Пусть A, B – объекты, $A = (S_A, M_A, s_A)$, $B = (S_B, M_B, s_B)$. Пусть (k_A, m_A) O M_A, (k_B, m_B) O M_B и в алгоритме m_A встретилась команда посылки сообщения k_B объекту B. В результате выполнения этой команды виртуальная машина передаст управление первой команде алгоритма m_B.

3. Команда возврата результата. Пусть $A = (S_A, M_A, s_A)$, $B = (S_B, M_B, s_B)$, (k_A, m_A) O M_A, (k_B, m_B) O M_B. Пусть объект A посылал сообщение k_B объекту B и в алгоритме m_B встретилась команда возврата результата. После выполнения этой команды состояние объекта A изменится с $s_A(t)$ на $s_A(t + 1)$, причём это изменение зависит от $s_A(t)$ и $s_B(t)$.

4. Команда порождения объекта. Пусть $C = (S_C, M_C, s_C)$ и (k_C, m_C) O M_C. Пусть команда порождения объекта встретилась в алгоритме m_C. В результате выполнения этой команды состояние $s(t)$ изменится на $s(t + 1)$, а также конструируется новый объект $O = (S, M, s)$, где S, M, s зависят от $s_C(t)$ – состояния объекта C в текущий момент времени.

Детали в этом определении уточняются в конкретной объектно-ориентированной модели вычислений (языке программирования). Более детальные определения объектно-ориентированного метода разработки программного обеспечения можно прочитать в [17], подробнее об ООП – в [18].

Одной из основных особенностей реальных объектно-ориентированных систем является то, что множества S и M не задаются для каждого объекта в отдельности, а указываются в специальных объектах-классах. Существует механизм наследования, иерархически упорядочивающий особые объекты (классы) в иерархическую структуру (в более сложном случае – в ациклический ориентированный граф). При этом, если объект O был порождён объектом-классом C_n и в иерархии наследования $C_n \to C_{n-1} \to \ldots \to C_1$ – путь от C_n к корню, то множество M объекта O есть объединение множеств методов, указанных в классах $C_1, \ldots, C_{n-1}, C_n$.

Подобные правила определения соответствия алгоритма селектору называются *метаобъектным протоколом* [19]. Метаобъектный протокол является одним из средств реализации АОП. Основой такой реализа-

ции является возможность поставить программный обработчик отправки определённого сообщения от определённого объекта к другому определённому объекту.

Если два аспекта пересекаются по одному программному модулю (методу некоторого объекта), то главный аспект реализуется объектно-ориентированным методом, а прочие аспекты реализуются методом перехвата посылки сообщения и запуска соответствующего алгоритма. Таким образом достигается раздельная реализация различных аспектов.

Как правило, для реализации такого перехвата вводится понятие метакласса. С одной стороны, метакласс – это класс класса как объекта (т.е. именно он ответственен за реализацию статических член-данных и член-функций в терминах языка C++), а с другой стороны, он управляет диспетчеризацией сообщений, пересылаемых объектам, являющимся экземплярами объекта-класса, который, в свою очередь, является экземпляром метакласса.

В случае, если транслятор с языка программирования переводит программу на некоторый промежуточный язык, выполняемый виртуальной машиной, подобный перехват сообщений реализуется средствами виртуальной машины. Таким образом, АОП реализовано в [15, 16], а также в языке AspectTalk. Однако существуют методы реализации АОП для компилируемых языков программирования.

3. Язык AspectTalk и пример реализации политики безопасности на этом языке

За основу языка AspectTalk взят язык Smalltalk [20, 21], к которому добавлен механизм метаклассов, аналогичный [15], с некоторыми дополнительными возможностями. А именно, множество объектов, к которым может быть произведено обращение из алгоритмов-перехватчиков, включает как объект, которому было послано сообщение, так и объект, пославший сообщение. Таким образом можно производить двустороннюю аутентификацию, что является необходимым условием реализации политик безопасности.

На листинге 3 показана программа, моделирующая поведение пользователей в системе. В строке 1 добавляются новые переменные alice и bob, в строках 2 и 3 они инициализируются объектами типа Process. В строках 4 и 6 эти объекты-процессы создают одноимённые файлы, а в строках 5 и 7 помещают в них свои конфиденциальные данные. Далее, в строках 8 и 9 alice пытается прочитать свой файл, а в строках 10 и 11 то же самое пытается сделать bob. Наконец, в строках 12 и 13 alice пытается прочитать файл bob'a.

```
 1 self addVariables: #(#alice #bob).
 2 alice <- Process new.
 3 bob <- Process new.
 4 alice execCreate: 'alice.txt'.
 5 alice execWrite: 'alice.txt' with: 'alice''s info'.
 6 bob execCreate: 'bob.txt'.
 7 bob execWrite: 'bob.txt' with: 'bob''s info'.
 8 ('Alice reads her file: ',
 9   (alice execRead: 'alice.txt')) writeLine.
10 ('Bob reads his file: ',
11   (bob execRead: 'bob.txt')) writeLine.
12 ('Alice reads Bob''s file: ',
13   (alice execRead: 'bob.txt')) writeLine
```

Листинг 3. Моделирование действий пользователей на языке AspectTalk

Объём статьи не позволяет дать полный текст программы. Отметим только, что класс Process является экземпляром метакласса ProcessClass, а класс File является экземпляром метакласса FileClass. Отдельно выделяются класс и метакласс системы (System и SystemClass соответственно).

В листинге 4 показан один из возможных вариантов определения указанных метаклассов. Такие метаклассы будем называть пустыми, поскольку они не несут никакой семантической нагрузки.

```
1 self addVariables: #(
2   #OwnerClass
3   #ThingClass
4   #SystemClass
5 ).
6 OwnerClass <- Metaclass new.
7 ThingClass <- Metaclass new.
8 SystemClass <- Metaclass new
```

Листинг 4. «Пустые» метаклассы на языке AspectTalk

Если исполнить программу в таком виде, как она описана выше, то можно увидеть, что alice имеет возможность успешно прочитать файл процесса bob.

Вывод программы на экран показан в листинге 5.

```
Alice reads her file: alice's info
Bob reads his file: bob's info
Alice reads Bob's file: bob's info
```

Листинг 5. Вывод программы, использующей «пустой» аспект

В листинге 6 показан аспект безопасности данной системы, реализованный на языке AspectTalk. Предполагается, что после объединения этого аспекта с главным аспектом программы прочитать данные, хранящиеся в каком-либо файле, сможет лишь пользователь, создавший этот файл.

В строках 1 – 7 объявляются переменные, используемые при работе алгоритма, а в строках 8 и 9 две из них – глобальные переменные, предназначенные для хранения идентификаторов, – инициализируются нулями.

В строках 10 – 18 определяется метакласс OwnerClass: его задача – в перехвате сообщения init, посылаемого каждому объекту при его создании (т.е. соответствующий алгоритм является конструктором объекта). В строке 13 выполняется вызов оригинального конструктора, а после к объекту добавляется член-данное id и инициализируется некоторым уникальным значением (хранящимся в переменной currentId).

В строках 19 – 24 определяется метакласс SystemClass: его задача – при обращении всякого объекта к нему, с целью выполнения действий над файлами, записать идентификатор этого объекта в глобальную переменную processId. Будут перехвачены только сообщения, селектор которых удовлетворяет регулярному выражению (read|write|create)File.* (строка 20). Непосредственно запись идентификатора происходит в строке 22, а в строке 23 вызывается оригинальный алгоритм для перехваченного сообщения.

```
1 self addVariables: #(
  2   #processId
  3   #currentId
  4   #OwnerClass
  5   #ThingClass
  6   #SystemClass
  7 ).
  8 processId <- 0.
  9 currentId <- 0.
 10 OwnerClass <- Metaclass new;
 11   addEnvelope: 'init' usingBlock: [
 12     | :sender :message :args |
 13     receiver call: message with: args.
 14     receiver addVariable: #id.
 15     currentId <- currentId + 1.
 16     id <- currentId.
 17     receiver addMethod: #id usingBlock: [ ^ id ]
 18   ].
 19 SystemClass <- Metaclass new;
 20   addEnvelope: '(read|write|create)File.*' usingBlock: [
 21     | :sender :message :args |
 22     processId <- sender id.
 23     receiver call: message with: args
 24   ].
 25 ThingClass <- Metaclass new;
 26   addEnvelope: 'init' usingBlock: [
 27     | :sender :message :args |
 28     receiver call: #init with: args.
 29     receiver addVariable: #id.
 30     id <- processId
 31   ];
 32   addEnvelope: '(read|write).*' usingBlock: [
 33     | :sender :message :args result |
 34     result <- 'This is NOT YOUR file!'.
 35     [ processId = id ] ifTrue: [
 36       result <- receiver call: message with: args
 37     ].
 38     ^ result
 39   ]
```

Листинг 6. Аспект безопасности, описанный на языке AspectTalk

В строках 25 – 39 определяется метакласс ThingClass: его задача – добавить идентификатор ко всякому объекту, соответствующему этому метаклассу (это производится в перехватчике метода init в строках 26 – 31). Как видно, идентификатор объекта становится равен идентификатору, хранящемуся в переменной processId, заполненной метаклассом SystemClass при обращении процесса к системе с просьбой создать файл (сообщение #createFile:).

В строках 32 – 39 определяется обработчик сообщений на чтение и на запись (соответствующее регулярное выражение (read|write).* указано в строке 32). В нём производится проверка идентификатора объекта-процесса, обратившегося к системе с просьбой о чтении или записи, и идентификатора объекта-файла. В случае совпадения идентификаторов производится запрашиваемое действие, а в случае несовпадения возвращается строка 'This is NOT YOUR file!'.

Вывод программы, полученной объединением главного аспекта и аспекта безопасности, показан в листинге 7.

```
Alice reads her file: alice's info
Bob reads his file: bob's info
Alice reads Bob's file: This is NOT YOUR file!
```

Листинг 7. Вывод программы, использующей аспект безопасности

Стоит отметить также сходство языка AspectTalk и языка MetaclassTalk, описанного в [16], несмотря на то, что их разработка производилась независимо. Так же, как и MetaclassTalk, язык AspectTalk обладает следующими преимуществами над языком AspectJ.

1. Возможность повторного использования аспектов в других приложениях.
2. Унифицированное определение статических конструкций аспекта (например, состав объектов – их член-данные, отношения наследования и агрегации) и динамических конструкций (перехват сообщений).
3. Возможность использования многих аспектов в одном приложении, пересекающихся по одному и тому же модулю.

Детальный анализ данного сравнения можно найти в [16].

Заключение

Аспектно-ориентированное программирование – одна из главных современных парадигм программирования. Она вводит уровень модульности, позволяющий изолированно разрабатывать различные аспекты приложения. Реализация политик безопасности – важный аспект современных компьютерных систем. АОП – это инструмент, позволяющий реализовывать политики безопасности удобным для человека образом. Такой инструмент в составе языка программирования AspectTalk, виртуальной машины и транслятора с AspectTalk на язык этой машины создан автором данной работы.

Грамматика языка AspectTalk включает в себя как объектно-ориентированные конструкции, которые используются при реализации основных частей приложений, так и аспектно-ориентированные конструкции, необходимые для реализации аспектов приложений. Программы, написанные на нем, после трансляции объединяются в одно целое и исполняются виртуальной машиной.

Язык AspectTalk – объектно-ориентированный и аспектно-ориентированный язык программирования, обладающий некоторыми преимуществами перед современными аспектно-ориентированными средствами разработки. Основой для AspectTalk послужил объектно-ориентированный язык программирования Smalltalk [20, 21], аспектно-ориентированное программирование реализуется с помощью метаобъектного протокола, аналогичного таковому в языке Python [15]. Язык AspectTalk схож с языком MetaclassTalk, описанным в [16], несмотря на их независимую реализацию. Оба языка обладают определёнными преимуществами над AspectJ в плане удобства реализации аспектов.

Поскольку аспектно-ориентированное программирование является перспективным способом реализации политик безопасности, исследования в области АОП языков и методов разработки способны вывести внедрение политик безопасности в компьютерные системы на качественно новый уровень.

ЛИТЕРАТУРА

1. *Department* of Defense Trusted Computer System Evaluation Criteria. DoD 5200.28-STD, 1985.
2. *Девянин П.Н.* Модели безопасности компьютерных систем: Учеб. пособие для студ. высш. учеб. заведений. М.: Издательский центр «Академия», 2005. 144 с.
3. *Таненбаум Э.* Современные операционные системы. 2-е изд. СПб.: Питер, 2005. 1038 с.
4. *Dijkstra E.W.* Selected Writings on Computing: A Personal Perspective. N.Y.: Springer Verlag, 1982. P. 60 – 66.
5. *Elrad T.*, *Aksit M.M.*, *Kiczales G.*, *et al.* Discussing Aspects of AOP // Communications of ACM. 2001. October. V. 44. No. 10. P. 33 – 38.
6. *Booch G.* Through the Looking Glass, 2001. http://www.ddj.com/architect/184414752.
7. *Elrad T.*, *Filman R.E.*, *Bader A.* Aspect-Oriented Programming // Communications of ACM. 2001. October. V. 44. No. 10. P. 29 – 32.

8. Язык программирования AspectJ. http://eclipse.org/aspectj.
9. *Diaz Pace J.A.*, *Campo M.R.* Analyzing the Role of Aspects in Software Design // Communications of ACM. 2001. October. V. 44. No. 10. P. 67 – 73.
10. *Lieberherr K.*, *Orleans D.*, *Ovlinger J.* Aspect-Oritnted Programming with Adaptive Methods // Communications of ACM. 2001. October. V. 44. No. 10. P. 39 – 41.
11. *Bergmans L.*, *Aksit M.* Composing Crosscutting Concerns Using Composition Filters // Communications of ACM. 2001. October. V. 44. No. 10. P. 51 – 57.
12. *Kiczales G.*, *Hilsdale E.*, *Hugunin J.*, *et al.* Getting Started with AspectJ // Communications of ACM. 2001. October. V. 44. No. 10. P. 59 – 65.
13. *Kiczales G.*, *Lamping J.*, *Menhdhekar A.*, *et al.* Aspect-Oriented Programming // Lecture Notes in Computer Science / Ed. by M. Aksit, S. Matsukoa. N.Y.: Springer Verlag, 1997. June. V. 1241. P. 220 – 242.
14. *Aspect*-Oriented Software Development. http://aosd.net.
15. Язык программирования Python. http://python.org.
16. *Bouraqadi N.*, *Seriai A.*, *Leblanc G.* Towards unified aspect-oriented programming // ESUG 2005 Research Conference. Brussels, Belgium, 2005. 22 p.
17. *Куликов М.Л.*, *Ромашкин Е.В.*, *Стефанцов Д.А.* Разработка средств моделирования политик безопасности операционных систем // Вестник ТГУ. Приложение. 2007. № 23. С. 189 – 193.
18. *Budd T.* An Introduction to Object-Oriented Programming. 3rd edition. Addison-Wesley, 2001. 648 p.
19. *Kiczales G.* The Art of Meta-Object Protocol. The MIT Press, 1991. 345 pp.
20. *Goldberg A.*, *Robson D.* Smalltalk 80, volume 1 – The Language and its implementation. Addison-Wesley, 1983.
21. *Budd T.* A Little Smalltalk. Addison-Wesley, 1987. 280 p.