

Барабанов А. Б.

Введение в системное программирование. Часть 4. Математическая модель.

Примечание.

В силу ряда причин все дальнейшие публикации этой темы, скорее всего, будут производиться на персональном сайте [1]. Поскольку статьи являются результатом развивающейся практической работы, то возможны некоторые опечатки, а также терминологические или даже логические расхождения текущих публикаций с предыдущими. Корректировка ранее опубликованных текстов производиться не будет до полной сборки работы в единую статью.

В предыдущих частях данного цикла [1] задача автоматизации системного администрирования решалась интуитивно понятным образом и рассуждения велись от очевидного восприятия задачи. Попробуем к этому же вопросу подойти с чисто формальных позиций. Формальный подход позволит абстрагироваться от излишнего влияния технологических деталей, но одновременно с этим он несет угрозу размывания конечной цели среди нагромождения логических построений. Потому сформулируем цель работы еще раз, чтобы не потерять её.

Итак, целью является созданию системы накопления опыта администрирования информационных систем, для того чтобы в дальнейшем получить экономию сил и средств за счет вторичного использования найденных ранее решений. Необходимость в такой системе возникает из-за того, что очень часто даже в высокой степени формализованные сведения о системном администрировании не могут быть повторно использованы из-за недостаточной параметризации, или в следствие неполноты описания условий применения, или из-за недостатка инструментального воплощения, или по какой-то иной причине. Не говоря уже о том, что далеко не все процессы и решения администрирования были доведены до степени инструментального воплощения и до сих пор публикуются в виде инструкций на естественном языке. Для того чтобы выработать общий подход к формализации знаний об администрировании информационных систем предпримем построение математической модели обсуждаемых объектов с использованием элементов теории множеств и математической логики. Начнем этот процесс с построения математической модели самой информационной системы, которая собственно и является объектом системного администрирования. Применяемая методика заключается в том, что описание предметной области на естественном языке заменяется соответствующим описанием на языке формальном, что позволяет в дальнейших рассуждениях, не учитывая семантику реального объекта, ограничиться синтаксическими преобразованиями математической модели.

Информационные технологии, впрочем, как и математические методы, лишены возможности прямого и непосредственного взаимодействия с миром реальных физических объектов. И, хотя, в данном случае «реальная» сторона представляется как раз информационным объектом, поступим с ним (точнее с «ней», с информационной системой) так, будто данный объект является полностью материальным, в отличие от создаваемой виртуальной модели.

Договоримся о нотации. Будем использовать прописные буквы для обозначения множеств,

функций и формул, и строчные для обозначения переменных и элементов множеств, выделяя все это в тексте полужирным начертанием — **A**, **a** и так далее. Множество допустимых логических значений ограничим традиционным вариантом {истина, ложь}, или, как это и будет далее обозначаться, {1, 0}, или {t, f}, или {true, false} .

Все примеры и построения в настоящей работе выполнены на основе программного обеспечения, распространяемого по свободным лицензиям таким, как GPL и её варианты. Подобный подход позволил сократить затраты автора на проведение самой работы и, безусловно, это должно упростить всем желающим заимствование полученных результатов. Однако, следует заметить, что все, далее описанное, с равным успехом применимо и к системам, основанным на проприетарном программном обеспечении.

Модель информационной системы.

Представим реальную информационную систему **Z** (Рис. 1), как множество областей {z}, доступных для обозрения и воздействия с внешними субъектами. В роли субъектов могут выступать или операторы — системные администраторы, или управляющие процессоры других информационных систем, выполняющие администрирующую задачу. Каждый раз, когда субъект обращается (обозревает) информационную систему **Z**, он рассматривает выбранную область z_i и получает (узнаёт) некую информацию о системе x_i . Можно сказать, что процедура обращения к системе **Z** может быть описана с помощью функции **Z(z)**, возвращающей значение x , сопоставленное с областью z внутри системы **Z**:

$$Z(z)=x \quad (\Phi-1)$$

Функцию **Z(z)** можно считать проекцией множества известных и дискретных областей {z} на состояние реальной системы **Z**.

Например, для оператора — человека такое обращение к различным областям можно описать на естественном языке как ряд повествовательных фраз следующим образом:

- *ip-адрес сетевого интерфейса есть ...;*
- *FQDN хоста есть ...;*
- *релиз операционной системы есть ...;*
- *FQDN почтовой системы есть ...*

В месте заключительного троеточия производится непосредственное обращение оператора к реальной системе для получения сведений об интересующем свойстве — области.

Если в качестве субъекта выступает внешний управляющий процессор, то аналогичное обращение можно выразить в терминах предложений языка программирования, которые после выполнения вернут соответствующую информацию об обозреваемой системе:

- *\$(sudo /sbin/ifconfig eth0 | grep "inet " | awk '{print \$2}' | cut -c 6-);*
- *\$(hostname);*
- *\$(cat /etc/redhat-release);*
- *\$(sudo /usr/sbin/postconf myhostname | awk '{print \$3}') ...*

И в том и в другом случае, возвращаемый набор значений предполагается одинаковым, если изучается одна и та же реальная система, например:

- *192.168.0.1;*
- *server.factory.localnet;*
- *CentOS release 5.5 (Final);*
- *mail.example.com ...*

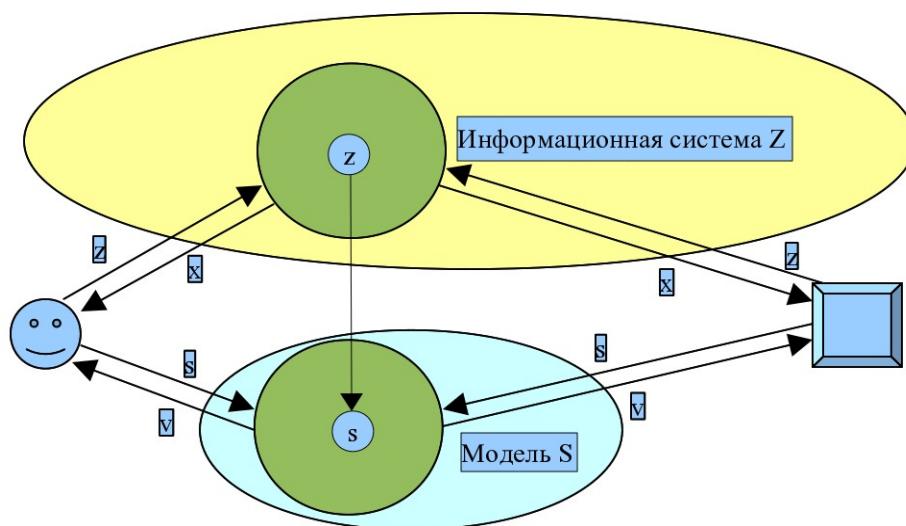


Рис. 1. Создание модели в процесс наблюдения реальной системы.

Здесь важно подчеркнуть, что реальный объект фактически не доступен для наблюдателей вне рамок выбранного способа взаимодействия, и потому такой способ восприятия Мира оставляет возможность несовпадения представлений субъекта и объективной реальности. Для примера, наличие файла /etc/redhat-release не превращает систему в семейства RedHat, и такой файл может находиться даже внутри Windows-среды. Однако, в рамках рассматриваемого подхода, считается, что каждое отдельное наблюдение достоверно квалифицирует изучаемый объект.

Изначально, информационная система Z , как материальный объект, уже содержит внутри себя все допустимое многообразие областей z , и все возможные связанные с ними значения x . Конкретный состав областей и значений определяется историей развития, способом построения и характером применения реального объекта, и для каждой информационной системы свой. Но до тех пор, пока внешний субъект не изучит систему, в ходе выполнения процедуры, соответствующей функции $Z(z)$, отображающей множество областей $\{z\}$ на множество сопоставленных данных $\{x\}$, её реальное состояние не известно.

Получение информации о системе Z может происходить, как в процессе её обследования (как ручного, так и автоматизированного), так и во время выполнения процедур настройки. И каждое выполнение функции $Z(z)$ приводит к постепенному пополнению коллекции известных о системе Z сведений. Назовем эту коллекцию, или множество, моделью, и обозначим символом S . Модель S имеет точно такие же свойства, или содержит такие же области, как реальная система Z . Она состоит из конечного и счетного множества областей $\{s\}$, которым сопоставлены известные значения, образующие множество $\{v\}$. Отображение областей модели S на множество связанных значений производится с помощью функции $S(s)$:

$$S(s)=v \quad (\Phi-2)$$

Инструментально модель может быть представлена тривиальной табличной структурой данных, в которой попарно связываются значения s_i и v_i . Иначе говоря, функция отображения для модели S вырождается в операцию доступа к элементам массива данных. Например, так:

$$S[s]=v \quad (\Phi-3)$$

Модель S может быть сформирована и без взаимодействия с реальной системой Z . Но тогда

подобная модель рассматривается как проект или техническое задание. Здесь в обсуждении появляется первая предикатная функция. Определим функцию проверки соответствия модели реальной системе $C(s)$:

$$C(s) = (Z(s) \equiv S(s)) \quad (\Phi-4)$$

Что означает существование в реальной системе Z и в модели S области s , и равенство связанных с этой областью значений, как в модели, так и в реальной системе. Область s_i для которой функция $C(s_i)$ истинна, назовем истинной областью модели для системы Z . Введенная предикатная функция везде будет употребляется в контексте некоторой модели S и реальной системы Z . И, строго говоря, следует в её написании указывать данную зависимость, приводя Z в списке параметров (элемент множества s уже указан). Но для удобства, если подразумевается, что рассматривается только одна реальная система, будет указываться или область s или все множество S , если обсуждаются разные модели или разные состояния одной и той же модели. Итак, уберем из полученного выражения обращение $S(s)$ и заменим его сразу проверяемым значением:

$$C(s) = (Z(s) \equiv v) \quad (\Phi-5)$$

Именно эту предикатную форму (функцию от двух аргументов) можно без труда преобразовать в высказывание на естественном языке, подставляя нужные значения в языковую форму « s есть v ». Например, для перечисленных выше областей:

- ip-адрес сетевого интерфейса *есть* 192.168.0.1;
- FQDN хоста *есть* server.factory.localnet;
- релиз операционной системы *есть* CentOS release 5.5 (Final);
- FQDN почтовой системы *есть* mail.example.com ...

На языке программирования все будет также просто:

- `$(sudo /sbin/ifconfig eth0 | grep "inet " | awk '{print $2}' | cut -c 6-) == "192.168.0.1";`
- `$(hostname) == "server.factory.localnet";`
- `$(cat /etc/redhat-release) == "CentOS release 5.5 (Final)";`
- `$(sudo /usr/sbin/postconf myhostname | awk '{print $3}') == "mail.example.com" ...`

Если модель образуется в процессе изучения оригинальной системы, то очевидно, что множество S является подмножеством Z :

$$S \subset Z \quad (\Phi-6)$$

Однако, если S представляет проект, то можно лишь утверждать о некотором пересечении данных множеств и не более. В этом случае возможны и такие ситуации, когда в формулу $\Phi-5$ будет подставляться значение области s , вообще не имеющее соответствующего эквивалента в множестве областей оригинала Z , что также будет приводить к возврату ложных значений функцией проверки соответствия модели реальной системе.

Функция $C(s)$ может возвращать истинные значения только в области пересечений множеств Z и S . Области s для которых функция $C(s)$ истинна образуют множество C :

$$C = \{ \forall s | Z(s) \equiv v \} \quad (\Phi-7)$$

Если множество C совпадает с множеством всех областей модели S , то модель S является истинным или достоверным представлением реальной системы Z . Для случая, когда модель S

представляет техническое задание на проведение работ по настройке целевой информационной системы, истинной модель станет только в случае успешного выполнения такого задания.

Предложенный путь построения модели полностью повторяет процесс познания и потому является тривиальным. Функции $Z(z)$ и $S(s)$ моделируют процесс получения знаний о состоянии системы Z и её модели S соответственно. Можно сказать, что невозможность выполнения соответствующей функции равносильна достижению границы возможного познания объекта. С учетом того, что информационная система является продуктом человеческого труда, она вся доступна для наблюдения и, значит, функция $Z(z)$ всегда существует и причем тривиальная. То же верно и в отношении модели S и её функции доступа $S(s)$. Тем не менее, если множество S конечно в следствие способа его получения, то множество Z можно считать бесконечным (хотя оно и отражает материальный объект) в силу того, что нет возможности определить его разбиение на области z априорно. Итак, функция $Z(z)$ относится к уровню реализации и поскольку $\Phi-7$ не содержит внутри себя иных функций, это дает основания считать $C(s)$ атомарной с точки зрения создаваемой модели.

На настоящем этапе рассуждений можно определить задачу администрирования, как согласование заданной модели S и реальной системы Z . Как именно, обсудим далее. Но ограничивать концепцию системного администрирования только рамками созданной модели S еще рано. Описанные абстракции формируют модель, но никак не гарантируют её логической адекватности. Например, в рамках полученной модели совершенно безразлично, считается ли адресом сетевого интерфейса 192.168.0.1, или строка «fooJaerb», или число 3.14! Кроме того, очевидно, что в реальной системе отдельные наборы областей наблюдения взаимосвязаны в силу функционального назначения самой информационной системы. Для того чтобы отразить семантику объекта наблюдения понадобится внести в модель дополнительные синтаксические языковые элементы.

Настройки информационной системы.

Определим дополнительное отношение d , которое назовём доменом, на группе областей $\{s\}$ из множества S модели информационной системы. Для этого сгруппируем области по их принадлежности к сервисным подсистемам. Например, в перечне областей:

- *ip-адрес сетевого интерфейса;*
- *FQDN хоста;*
- *релиз операционной системы;*
- *FQDN почтовой системы...*

первую и четвертую можно отнести к домену, соответствующему почтовому сервису **postfix**. Обозначим множество всех доменов модели S буквой D . Домены — множества могут пересекаться. Так область, названная «*ip-адрес сетевого интерфейса*» будет входить во все домены, отвечающие за сетевые службы. Каждая область, принадлежащая множеству S , должна принадлежать хоть какому-то домену:

$$D = \{d | d \subset S\} \quad (\Phi-8)$$

Сделаем замечание на счет множества всех доменных множеств $\{d\}$. Можно эти множества представить, как непустые конечные последовательности символов $\{s\}$ из фиксированного конечного списка S . Тогда, согласно утверждению [2, стр.210, принцип (A)] данное множество всех доменов $\{d\}$ счетно. Счетность множества подразумевает априорную его упорядоченность, что будет использовано в дальнейших рассуждениях. То же самое верно и в отношении множества областей, составляющих домен: это множество также счетно и может быть упорядочено.

С описываемым отношением D свяжем одноименную предикатную атомарную функцию $D(\dots)$. Валентность этой функции будет соответствовать числу областей включенных в домен. Для краткости обращение к данной функции будем записывать с нужным для обсуждаемого контекста числом аргументов, например, $D^n(s_1, s_2)$, подразумевая, что на самом деле функция применяется ко всему множеству областей, составляющих домен, или в форме $D(d)$, указывая тем самым, что аргументом является множество областей, составляющих домен d , или в форме $D(S)$, предполагая, что все аргументы принадлежат модели S . Функция возвращает логическое значение истина или ложь в зависимости от успеха выполнения соответствующей сервисной подсистемы в реальной системе Z (здесь тоже получается, что Z является неявным аргументом, и опускается в списке аргументов до тех пор пока реальная система Z подразумевается контекстом исполнения функции). Проверить это можно, например, следующим образом. Для домена **postfix** функция D_{postfix} вернет истину (или **t**, или 1) в том случае, если проверка возможности рестарта smtp-сервера Postfix «`sudo /sbin/service postfix restart`» будет успешной, иначе вернется значение ложь (или **f**, или 0).

Все вышесказанное не исключает существования доменов, построенных на иной сервисной основе, например, домен **reboot**, связывающий те области модели, от сопоставленных значений которых зависит успех запуска всей информационной системы.

Домены определяют семантику внутренних взаимосвязей информационной системы и множество доменов $D=\{d\}$ фактически соответствует множеству потенциально возможных свойств системы и множеству возможных функций $\{D(s)\}$ на множестве областей, принадлежащих некоторой модели S информационной системы Z . Каждая функция, как сказано выше, может быть истинной (вычислимой) для определенного состояния системы, соответствующего набору проекций, составляющих домен, или нет — ложной.

Множество вычисляемых (истинных) функций $D(s)$ на множестве истинных областей некоторой модели S составляет настройку (множество настроек) данной информационной системы:

$$A = \{d | d \in D, D(d) \equiv 1\} \quad (\Phi-9)$$

Мощность множества настроек, $|A|$, определяет степень настройки системы. Здесь уместно заметить, что факт успешного запуска сервиса никак не указывает на особенности его настройки. Иначе говоря, тот же самый Postfix может иметь огромное число вариантов настройки, при которых он будет успешно стартовать, но совершенно по разному функционировать. В разрабатываемой математической модели все эти варианты не различимы, для оценки степени настройки, как мощности множества A . Далее степень настройки будем называть мощностью системы. Выполнение функций $D(s)$ в контексте одной лишь только модели S будет возвращать всегда истинное значение. Это надо понимать так, что модель предполагает, что если в реальной системе установить все перечисленные настройки, то все задействованные функциональные возможности будут работать. Поэтому, для модели S мощность множества настроек совпадает с мощностью множества доменов. Для её расчета в этом случае не надо напрягаться. Однако, применительно к реальной системе это уже не так. В этом случае следует учесть степень настройки и рассчитывать функцию $D(d)$ только для доменов, состоящих из истинных областей:

$$A = \{d | d \in D, D(d) \equiv 1, (\forall s | s \in d) C(s) \equiv 1\} \quad (\Phi-10)$$

Теперь уже можно более точно описать задачу администрирования, как согласование заданной модели S и реальной системы Z с целью достижения максимальной мощности настройки системы Z , выраженной в числе успешно работающих сервисов A .

Буквально трактуя эту рекомендацию определим следующую схему элементарной итерации

администрирования (Рис. 2), которую назовем формулой перехода P .

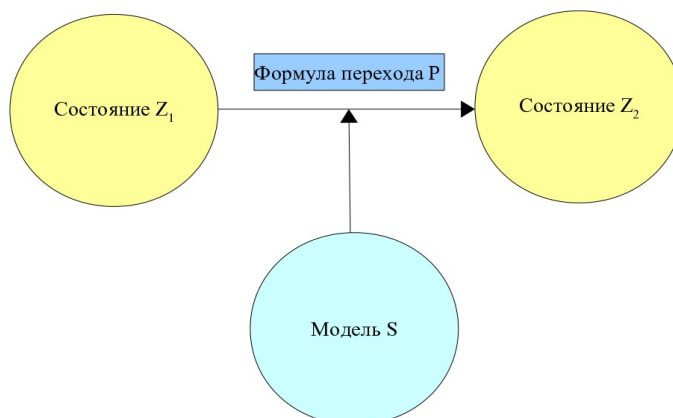


Рисунок 2. Элементарный шаг настройки информационной системы.

Под согласованием состояний понимается в данном случае перебор областей модели S и установление соответствующих значений в реальной системе Z . На рисунке 2 вся работа по согласованию скрыта в формуле перехода P . Рассмотрим её подробнее.

Изменение состояний информационной системы.

Прежде всего уточним, что понимается под состоянием информационной системы ДО выполнения формулы перехода P . Заметим (Рис. 2), что модель S в процессе администрирования статична и не меняет своего состояния. Предположим, что некий указатель перебирает все области s модели, и в некоторый момент времени t_i области включая s_{i-1} уже обработаны, и на данном шаге должна обрабатываться область s_i . Это значит, что внутри информационной системы Z_1 , множество областей $S_1 = \{s_0, s_1, \dots, s_{i-1}\}$ уже присутствуют и имеют истинные (совпадающие с моделью) значения:

$$Z_1 = \{Z \setminus S_1, S_1\} \quad (\Phi-11)$$

Выполнение формулы перехода должно произвести согласование следующей по счету области s_i так, чтобы в конце концов она тоже стала истинной:

$$Z_2 = \{Z \setminus \{S_1, s_i\}, S_1, s_i\} \quad (\Phi-12)$$

Следовательно, формула перехода должна содержать внутри функцию изменения состояния области реальной системы в соответствие с настройками, указанными в модели. Исходя из тех же соображений, что данная функция имеет смысл только в контексте совершенно определенной реальной информационной системы Z , не будем включать последнюю в список параметров и ограничимся одновалентной формой задания функции модификации $F(s)$. Безусловно, это должна быть предикатная и атомарная с точки зрения модели функция. Но вот результат её вызова будет немного не таким, как можно ожидать. Пусть эта функция возвращает истину в случае синтаксической корректности её реализации. Причины такого решения будут объяснены позже. В принципе, одной этой функции вполне достаточно, чтобы выполнить поставленную задачу. Но если все алгоритмы изменения состояний описывать только с помощью функции $F(s)$, то они будут слишком трудоемкими, так как это приведет к

тому, что все действия по настройке придется выполнять вне зависимости от реального состояния системы. Чтобы избежать подобного как раз и понадобится функция проверки состояния $C(s)$.

В первом приближении формула для $P(s)$, назовем её пока $P^0(s)$, может быть записана так:

$$P^0(s) = C(s) \vee F(s) \quad (\Phi-13)$$

Полученный дизъюнкт не следует правилу коммутативности и не допускает перестановки своих термов. Поэтому, чтобы не создавать сложных исключений, сразу удалим из применяемой аксиоматики закон коммутативности для операции \vee для нашей математической модели.

Однако, вспомним, что сама функция изменения состояния $F(s)$ никак не отмечает факт успешного достижения цели — модификации области системы Z . И результат её применения может быть ошибочным даже в случае процедурного успеха. С учетом сказанного поправим формулу изменения состояния очевидным способом:

$$P^0(s) = C(s) \vee (F(s) \wedge C(s)) \quad (\Phi-14)$$

Функция $P^0(s)$ будет истинной в том случае, если условие истинности области модели будет соблюдаться или «до» вызова $P^0(s)$, или «после» её выполнения. Здесь становится очевидным, что конъюнкция, применительно к создаваемой модели, тоже не может быть коммутативной, иначе в чем смысл менять состояние после проверки успешности применения изменений.

Теперь, вспомним, что все области распределяются по соответствующим доменам. Функция $D(s)$, в отношении обсуждаемого реального объекта ИС, вычисляя свое значение производит рестарт реального сервисного процесса (а иначе как еще узнать, будет ли он работать?). И, значит, что выработанную выше формулу следует расширить с учетом того, в случае модификации некоторой области s , надо автоматически пересчитать вычислимость всех доменов, которые включают модифицированную область — а, вдруг, они станут истинными! Записать это можно следующим образом:

$$P^0(s) = C(s) \vee (F(s) \wedge C(s) \wedge (\forall d | s \in d) D(s)) \quad (\Phi-15)$$

Таким образом, при рассмотрении процесса элементарного изменения, внутри формулы появляется конструкция «для всех функций $D(s)$, включающих в список аргументов область s », квантифицирующая отношение областей. Тем самым предполагается, что строгое математическое описание модели информационной системы требует применения логики второго порядка.

Пришло время учесть, что, собственно, результат применения $F(s)$ совсем не обязательно будет успешным. Что это значит для реальной информационной системы? Это значит, что после неудачного изменения потребуется выполнить «откат» к прежним значениям. Так возникает необходимость в еще двух атомарных предикатных функциях $B(s)$ и $R(s)$, первая из которых запоминает состояние изменяемой области, а вторая восстанавливает ранее запомненное значение. С учетом вышесказанного получится следующая формула изменения состояния:

$$P^0(s) = C(s) \vee (B(s) \wedge F(s) \wedge ((C(s) \wedge (\forall d | s \in d) D(s)) \vee (R(s) \wedge (\forall d | s \in d) D(s)))) \quad (\Phi-16)$$

Такую формулу уже затруднительно и произнести, и понять. Она явно требует упрощения. Начнем с учета факта, что данная формула применяется последовательно ко всем областям

модели, и, следовательно, так или иначе «перебирает» все множество доменов, входящих в S . Поскольку, как было замечено выше, множество доменов можно упорядочить, то для каждого конкретного изменения состояния достаточно учитывать его включение не более, чем в один домен. Случаи идентичности доменов (совпадения множества областей, их составляющих) следует рассматривать также, как один домен, что в принципе легко решается на уровне программной реализации. Таким путем избавляемся от квантификации доменов и получаем следующий вариант формулы изменения состояний:

$$P^0(s) = C(s) \vee (B(s) \wedge F(s) \wedge ((C(s) \wedge D(s)) \vee (R(s) \wedge D(s)))) \quad (\Phi-17)$$

Чтобы не пестрило в глазах от числа скобок, упраздним в нашей нотации указание аргумента s , так как он подразумевается контекстом:

$$P^0 = C \vee (B \wedge F \wedge ((C \wedge D) \vee (R \wedge D))) \quad (\Phi-18)$$

Опишем, что определяет полученная формула. Итак, изменение состояния начинается с проверки соответствия Z конечному состоянию (функция C), и если $Z(s)$ соответствует $S(s)$, то формулу можно считать выполненной и истинной. В противном случае выполняется сохранение состояния системы Z , для области s , и предпринимается попытка модификации Z в соответствие с требованиями модели S . После этого снова производится проверка соответствия системы Z заданной модели S , и в случае неудачи выполняется восстановление предыдущего состояния Z . Проверка истинности домена необходима в любом случае, как и при удачной модификации, так и в случае отката состояния.

Заметно, что здесь не рассматривается случай, когда сама функция $F(s)$ может завершиться неудачей, то есть вернет значение **false**, что также потребует действия, аналогичного неудачной модификации. Внесем и эту процедуру в формулу:

$$P^0 = C \vee (B \wedge (F \wedge ((C \wedge D) \vee (R \wedge D)) \vee (R \wedge D))) \quad (\Phi-19)$$

Итак, $\Phi-19$ исчерпывающе описывает один элементарный шаг преобразования Z в соответствие с требованиями S . Тогда полностью всю настройку можно описать как функцию настройки системы согласно модели $P(S)$:

$$P(S) = (\forall s | s \in S) P^0(s) \quad (\Phi-20)$$

По сути, это соответствует последовательному применению формулы перехода $P^0(s)$ ко всему множеству областей s , описанных в модели S , или, в принятой терминологии, это можно описать, как упорядоченное множество шагов администрирования:

$$P(S) = (\forall s | s \in S^n) P^0(s) = \{P^0(s_1), P^0(s_2), \dots, P^0(s_i), \dots, P^0(s_n)\} \quad (\Phi-21)$$

Или графически, в виде диаграммы на рисунке 3.

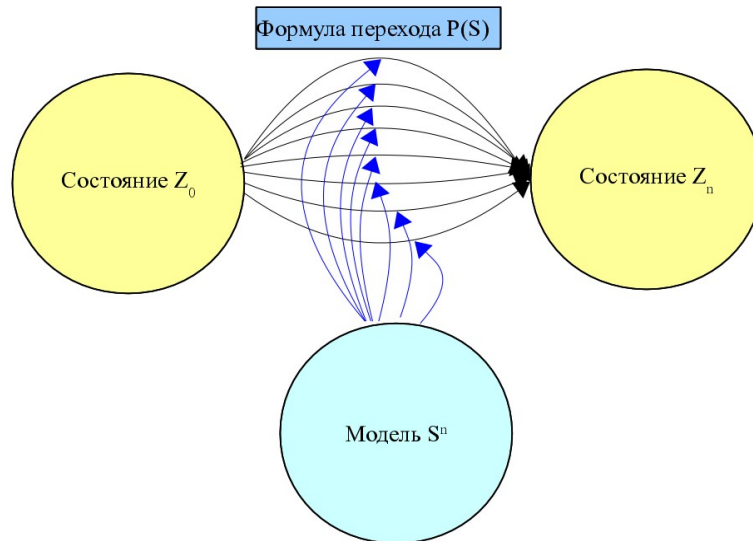


Рисунок 3. Применение формулы перехода ко всем областям модели S .

На рисунке 3 все формулы перехода выполняются параллельно. Строго говоря, это не противоречит природе информационных систем. Кроме того, в условиях создания математической модели тоже ничего на этот счет пока не устанавливалось. Однако, теперь надо сделать реалистический выбор и договориться, что все шаги по настройке системы будут выполняться строго последовательно, как на рисунке 4.

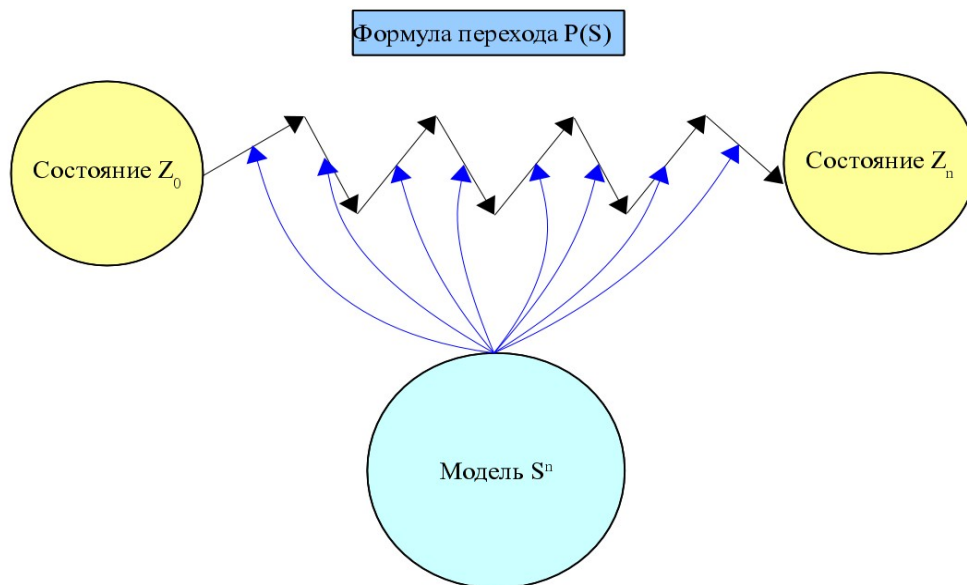


Рисунок 4. Последовательная настройка системы Z .

Но, ведь, каждый переход $P^0(s)$ представляется в реальности некоторым программным кодом, а в рассматриваемой модели логической формулой. И потому, очевидным ходом будет представить последовательность настройки системы Z конъюнктом, элементы которого соответствуют вычислению формулы перехода $P^0(s)$ для всех областей модели S :

$$P(S) = (\forall s | s \in S^n) P^0(s) = P^0(s_1) \wedge P^0(s_2) \wedge \dots \wedge P^0(s_i) \wedge \dots \wedge P^0(s_n) \quad (\Phi-22)$$

Что должно происходить в случае ошибочного исполнения некоторого очередного перехода состояния — ну, не получилось выполнить настройку в силу каких-то причин (или недостаток модели, или проблемы реализации, или иной фактор)? Попробуем рассуждать. Изначально, формула перехода $P^0(s)$ строилась так, чтобы учесть возможную ошибку при исполнении функции $F(s)$ (обсуждение Ф-14). Для этого и была введена повторная проверка $C(s)$. Снова рассмотрим формулу Ф-19. Функции $B(s)$ и $R(s)$, согласно замыслу, должны возвращать почти всегда истинное значение t (или 1), кроме случаев повреждения самой исполняющей системы или подобных. Подставим константу t вместо B и R в Ф-19 и упростим её:

$$P^0 = C \vee (t \wedge (F \wedge ((C \wedge D) \vee (t \wedge D)) \vee (t \wedge D))) \quad (\Phi-23)$$

$$P^0 = C \vee (F \wedge ((C \wedge D) \vee D) \vee D) \quad (\Phi-24)$$

В полученной формуле видно, что значение $P^0(s)$ зависит или от $C(s)$ или от $D(s)$. Если членами конъюнкта в Ф-22 будут формулы эквивалентные Ф-24, то как можно гарантировать корректность выполнения последовательной настройки в случае, если нет возможности увязать возвращаемое значение и фактический успех работы по настройке? Исправим это.

Во-первых, необходимо обеспечить возвращаемое значение **false** в случае неудачи в процессе настройки — это два последних обращения к функции $D(s)$ в Ф-24 или в Ф-19. Подмешать «& false» не кажется хорошим решением, так как, скорее всего, реализация (точнее, непосредственно кодирование) потребует создание специальной функции для завершения программы, которую назовем $X(s)$. Эта функция должна будет выполнить все необходимые действия по обработке ошибочного состояния (например, распечатать текст ошибки) и завершить процесс настройки, или непосредственно в ходе своего исполнения, или вернув значение **false** для прерывания вычисления конъюнкта Ф-22. Но и эту функцию просто «подмешать» тоже не получится, так как возможно **false** вернет вызов $D(s)$ и тем самым сделает исполнение $X(s)$, на которое мы уже так положились, невозможным. Придется воспользоваться несколько неуклюжим способом:

$$P^0 = C \vee (B \wedge (F \wedge ((C \wedge D) \vee (R \wedge (D \vee t) \wedge X)) \vee (R \wedge (D \vee t) \wedge X))) \quad (\Phi-25)$$

Просто удалить вызов $D(s)$ нельзя, так как он содержит необходимый для работы функционал. Вариант « $\vee true$ » одновременно и решит проблему, и позволит остаться в рамках теории исчисления высказываний. Аналогичным образом маскируем ошибку срабатывания $R(s)$:

$$P^0 = C \vee (B \wedge (F \wedge ((C \wedge D) \vee ((R \vee t) \wedge (D \vee t) \wedge X)) \vee ((R \vee t) \wedge (D \vee t) \wedge X))) \quad (\Phi-26)$$

Но и это еще не все. Во-вторых, в случае успешного внесения изменения в состояние обрабатываемой области необходимо обеспечить возвращение значения **true** формулой перехода $P^0(s)$ вне зависимости от результата пересчета вычислимости домена $D(s)$, для того чтобы вычисление Ф-22 не прерывалось. Снова обратим внимание на рисунок 2. Если считать, что в Z_1 область s не настроена, и, следовательно $C(s)$ ложна, а функция изменения состояния $F(s)$ с точки зрения исполняющей системы верна (правильно выполняется в реальной системе), то логическое значение, возвращаемое формулой перехода $P^0(s)$ будет определяться успешностью вычисления домена D . Здесь следует напомнить, что множества S и D счетные и упорядоченные. Это значит, что если допустить 100% правильность кодирования и исполнения функции изменения состояния $F(s)$, то последовательный перебор шагов настройки всех областей, входящих в домен, рано или поздно приведет к успешности

вычисления рассматриваемого домена **D**. Иначе говоря, на всех промежуточных шагах, кроме последнего, можно не только не учитывать значение, возвращаемое **D(s)**, но даже не производить сам пересчет вычислимости домена. Таким образом, вместо **P⁰(s)** для всех областей домена кроме последнего надо будет, например, использовать формулу **P¹(s)**, где первый вызов **D(s)** исключен:

$$(\forall s | s \in \{d^n \setminus s_{n_j}\}) P^1 = C \vee (B \wedge (F \wedge (C \vee ((R \vee t) \wedge (D \vee t) \wedge X)) \vee ((R \vee t) \wedge (D \vee t) \wedge X))) \quad (\Phi-27)$$

И лишь для совершения последней настройки, отвечающей за некоторый домен **D**, успешность вычисления домена **D(s)** будет совпадать с успехом всех модификации группы областей, принадлежащих домену **D**.

Вернемся к формуле $\Phi-26$. Попробуем её упростить. Сначала воспользуемся законом дистрибутивности ([2], страница 27, формула 35) для связки с функцией **F(s)**:

$$F \wedge ((C \wedge D) \vee ((R \vee t) \wedge (D \vee t) \wedge X)) = (F \wedge (C \wedge D)) \vee (F \wedge ((R \vee t) \wedge (D \vee t) \wedge X)) \quad (\Phi-28)$$

Получим:

$$P^0 = C \vee (B \wedge (F \wedge (C \wedge D) \vee F \wedge ((R \vee t) \wedge (D \vee t) \wedge X) \vee ((R \vee t) \wedge (D \vee t) \wedge X))) \quad (\Phi-29)$$

Затем, применим закон элиминации ([2], страница 27, формула 40) для исключения второго вызова все той же **F(s)**:

$$F \wedge ((R \vee t) \wedge (D \vee t) \wedge X) \vee ((R \vee t) \wedge (D \vee t) \wedge X) = ((R \vee t) \wedge (D \vee t) \wedge X) \quad (\Phi-30)$$

Получим:

$$P^0 = C \vee (B \wedge (F \wedge (C \wedge D) \vee ((R \vee t) \wedge (D \vee t) \wedge X))) \quad (\Phi-31)$$

Теперь раскроем скобки там, где это можно сделать:

$$P^0 = C \vee (B \wedge (F \wedge C \wedge D \vee (R \vee t) \wedge (D \vee t) \wedge X)) \quad (\Phi-32)$$

Снова используем закон дистрибутивности:

$$B \wedge (F \wedge C \wedge D \vee (R \vee t) \wedge (D \vee t) \wedge X) = (B \wedge F \wedge C \wedge D) \vee (B \wedge (R \vee t) \wedge (D \vee t) \wedge X) \quad (\Phi-33)$$

Посмотрим, что получилось:

$$P^0 = C \vee (B \wedge F \wedge C \wedge D) \vee (B \wedge (R \vee t) \wedge (D \vee t) \wedge X) \quad (\Phi-34)$$

Для преобразований использовались законы исчисления высказываний [2] с учетом ранее заявленной поправки о неприменимости коммутативности. Создание и обоснование полной аксиоматики математической модели информационной системы пока не входит в задачу данной работы. Потому, сейчас, как и в первом случае, просто обойдемся указанием на некоторое разумное исключение. В полученном дизъюнкте $\Phi-34$ второй вызов **B(s)** явно противоречит условиям предметной области. Поскольку вычисление $\Phi-34$ будет производиться строго последовательно (закон коммутативности не работает!), то второй вызов **B(s)** не сработает (не будет иметь смысла) по причине существования более ранней

копии изменяемых данных, оставленной первым обращением к этой функции. Потому, просто удалим его:

$$P^0 = C \vee (B \wedge F \wedge C \wedge D) \vee ((R \vee t) \wedge (D \vee t) \wedge X) \quad (\Phi-35)$$

Это окончательный вид формулы перехода состояния. Формула $\Phi-35$ полностью эквивалентна формуле $\Phi-26$. Желающие могут проверить это путем построения таблиц истинности для каждой из формул.

Соответственно, формула $P^1(s)$ будет иметь вид:

$$P^1 = C \vee (B \wedge F \wedge C) \vee (R \wedge (D \vee t) \wedge X) \quad (\Phi-36)$$

Итак, формулы $\Phi-35$ и $\Phi-36$ описывают элементарные операции, из которых состоит настройка областей, сгруппированных по принципу принадлежности к некоторой сервисной функции или иному критерию в абстрактный домен. Каждую такую операцию, соответствующую указанным формулам, далее будем называть «шагом», а всю совокупность «шагов» для настройки некоторого домена будем называть «процедурой».

Тогда вся последовательность настройки некоторой системы Z в соответствии с моделью S , будет состоять из перебора всех доменов системы S и обработки их в соответствии с сопоставленным этим доменам алгоритмическим «процедурам». Назовем такую последовательность исполнения процедур «решением» для модели S , обозначим функцией системного администрирования $A(S)$, и опишем следующим образом:

$$A(S) = (\forall d | d \subset S) ((\forall s | s \in \{d^n \setminus s_n\}) P^1(s) \wedge P^0(s_n)) \quad (\Phi-37)$$

Здесь снова возвращается квантор общности, применительно к доменам. Но в данном случае домены рассматриваются как подмножества S , а не как свойства элементов s — принадлежности к доменам. Иначе говоря, такое определение системного администрирования не выходит за рамки логики первого порядка. Вычисление или дальнейшее алгоритмическое разворачивание полученной формулы производится путем создания циклов или последовательностей операций в соответствии с кванторами и подстановкой параметров из исходной модели S .

На этом пока оставим теорию и рассмотрим возможные подходы к практической реализации алгоритмов, построенных в соответствии с предложенными формулами $\Phi-35$, $\Phi-36$ и $\Phi-37$.

Инструментальная реализация модели информационной системы.

Прежде всего, сведем использованные абстрактные конструкции — множества, функции и формулы, в единую таблицу 1.

№№	Название	Класс	Обозначение
1	Модель	Множество	$S = \{s\}$
2	Параметры модели	Множество	$V = \{v\}$
3	Домен	Множество	$d = \{s\}$
4	Доступ к реальной системе	Функция платформы	$Z(s)$
5	Доступ к модели	Функция платформы	$S(s)$
6	Проверка	Атомарная функция	$C(s)$

7	Настройка	Атомарная функция	$D(s)$
8	Изменение	Атомарная функция	$F(s)$
9	Резервное копирование	Атомарная функция	$B(s)$
10	Восстановление	Атомарная функция	$R(s)$
11	Прерывание исполнения	Функция платформы	$X(s)$
12	Изменение состояния	Формула	$P^0(s)$
13	Изменение состояния	Формула	$P^1(s)$
14	Администрирование	Формула	$A(S)$

Таблица 1. Средства описания математической модели системного администрирования.

Таблица 1 позволяет наглядно представить деление всех использованных средств на три категории. Первая категория составляет *множества*. Это множество элементов модели, вектор их параметров и множество доменов. Инструментальной реализацией множеств являются структуры данных, например таблицы СУБД. С точки зрения построения СУБД можно обойтись без задания специальных таблиц для доменов, заменив их на дополнительный атрибут в таблице элементов модели. Даже «не заменив», а «оставив лишь», поскольку принадлежность области домену так или иначе должна указываться в таблице описывающей саму область. Но это как раз тот самый случай объективно нужной «денормализации», с которой надо смириться, точно также, как пришлось прибегнуть к предположению лишь циклического обхода всех элементов проекции, принадлежащих домену, для избавления от квантификации свойств областей в предыдущем разделе.

Категория функций включает все необходимые низкоуровневые по отношению к рассматриваемой абстракции математической модели процедуры взаимодействия с реальным объектом. Функции разделяются на два подкласса. Во-первых, это «функции платформы», те что обеспечиваются выбранным инструментарием и способом, каким этот инструментарий может взаимодействовать с реальной системой — как он «может» её настраивать. Второй подкласс, так называемые «атомарные функции» или предикаты, возвращающие логические значения, и используемые для построения формул. Эти функции не только платформенно-зависимые, но еще и в существенной степени определяются еще более низкоуровневым разделением на классы самих областей модели. Правильно будет сказать, что эти функции представляют методы класса в зависимости от типа области модели. Таким образом в реализации возникает еще одна структура данных, которая задает код (или способ его генерации) всех перечисленных функций в зависимости типа исходной для модели среды.

Особняком стоит функция X , которая, хотя, и имеет характеристики предиката, тем не менее полностью зависит от выбранного инструментария реализации, так как введена искусственным образом для решения сугубо утилитарной задачи, прерывания работы исполняющей системы.

Последняя категория это формулы. Они определяют последовательность действий для решения задачи комплексной настройки системы Z согласно модели S , как техническому заданию. Формулы были созданы в процессе разработки математической модели, а потому они никак не связаны с дополнительной семантикой предметной области и их можно реализовать только за счет алгоритмов системы автоматизации администрирования. Фактически это и есть алгоритм генерации кода для выполнения административной задачи настройки информационной системы.

Данные из Таблицы 1 позволяют продемонстрировать построение модели на простом примере. Например, для области «FQDN почтовой системы». Определим все

взаимосвязанные с этой областью данные и функции, и запишем их в Таблицу 2.

Область s	FQDN почтовой системы
Сопоставленное значение v	mail.example.com
Функция проверки C(s)	sudo /usr/sbin/postconf myhostname awk '{print \$3}' grep \$v
Резервное копирование B(s)	sudo cp -f /etc/postfix/main.cf /etc/postfix/main.cf.orig
Восстановление R(s)	sudo test -r /etc/postfix/main.cf.orig && sudo cp -f /etc/postfix/main.cf.orig /etc/postfix/main.cf
Функция модификации F(s)	sudo /usr/sbin/postconf myhostname=\$v
Вычислимость домена D(s)	sudo /sbin/service postfix restart && sudo /sbin/service postfix status
Прерывание исполнения X(s)	echo ошибка при настройке \$s ; exit -1

Таблица 2. Пример реализации настройки.

Если сравнить первую и вторую таблицу, можно заметить, что функция **S(s)** во второй таблице никак не отражена. Дело в том, что назначение этой функции в инициализации самой модели, и, значит, можно сказать, что данная функция задается Таблицей 2.

Очевидно, все области, описывающие параметры, содержащиеся в файле /etc/postfix/main.cf будут определены аналогичным образом. Методически можно выделить специальный класс областей, взаимосвязанных с файлом настройки postfix и описать его например так, как приведено в Таблице 2, и тогда при задании каждого нового экземпляра этого класса необходимо будет лишь указать совсем малую часть данных — Таблица 3. Кстати, можно задать размещение файла настроек postfix с помощью запроса $\$(rpm -ql postfix | grep main.cf\$)$.

Класс	Параметр Postfix
Ключевой параметр функций	k
Функция проверки C(s)	sudo /usr/sbin/postconf \$k awk '{print \$3}' grep \$v
Резервное копирование B(s)	sudo cp -f /etc/postfix/main.cf /etc/postfix/main.cf.orig
Восстановление R(s)	sudo test -r /etc/postfix/main.cf.orig && sudo cp -f /etc/postfix/main.cf.orig /etc/postfix/main.cf
Функция модификации F(s)	sudo /usr/sbin/postconf \$k=\$v
Вычислимость домена D(s)	sudo /sbin/service postfix restart && sudo /sbin/service postfix status

Таблица 2. Пример определения функций класса.

Класс	Параметр Postfix
Область s	FQDN почтовой системы
Сопоставленное значение v	mail.example.com
Ключевой параметр k	myhostname

Таблица 3. Пример ООП реализации модели.

Цепочку инкапсуляций можно продолжить еще глубже, определив класс областей, которые относятся к текстовым файлам, воспользовавшись тем, что для них функции резервного копирования и восстановления будут однотипны и тогда класс «Параметр Postfix» можно будет создать на его основе. И так далее.

Приемы ООП в реализации подключаются самым естественным образом. Как будет продемонстрировано далее, упрощение решения задач системного администрирования за счет механизма «наследования» является одним преимуществ предлагаемого подхода. А пока представим простенький код прототипа системы автоматического администрирования. В качестве инструментальной основы выберем Ruby, с которым можно ознакомиться с использованием [3].

Создадим класс, как было предложено чуть выше, для определения класса настроек Postfix. Назовем его Postfixparams. Пусть он наследует методы B и R, соответствующие функциям **B(s)** и **R(s)** от родительского класса Fileloc, а методы C и F (функции **C(s)** и **F(s)**) определяет собственные. Например так (Листинге 1):

```
class Postfixparams < Fileloc
  # область представленная параметрической строкой в формате postfix main.cf
  def initialize (param0,value0)
    @@filename="/etc/postfix/main.cf"
    @param=param0
    @value=value0
  end
  def C # проверка
    "sudo /usr/sbin/postconf #@param | awk '{print $3}' | grep #@value >>/dev/null 2>&1"
  end
  def F # модификация
    "sudo /usr/sbin/postconf -e #@param=#@value"
  end
end
```

Листинг 1. Определение класса параметров postfix.

Тогда базовый класс Fileloc можно определить следующим образом (Листинг 2):

```
class Fileloc
  # область представленная файлом
  @@filename=""
  def B # резервное копирование
    "sudo /bin/cp #@@filename #@@filename.orig"
  end
  def R # восстановление
    "sudo test -r #@@filename.orig && sudo /bin/cp #@@filename.orig #@@filename"
  end
end
```

Листинг 2. Определение класса областей файлового хранения.

Теперь при создании класса настроек того же Postfix, хранящихся в иных файлах (например, для chroot-ой реализации), можно будет изменяя переменную класса @@filename получать

корректные методы B и R.

Увязать в одну цепочку наследования еще и домены не получится. И здесь придется воспользоваться имитацией множественного наследования, принятой в Ruby, за счет механизма включения модулей. Построим цепочку наследования: пусть модуль Postfixdomain, включаемый в обработку всех областей домена Postfix, сам строится с использованием модуля Servicedomain, который, в свою очередь, послужит основой для всех модулей, определяющих сервисные домены, службы которых управляются с помощью вызова /sbin/service (Листинг 3).

```
module Servicedomain
  # домен, определяемый отношением к сервисному процессу
  @@service=""
  def D # проверка вычислимости домена-сервиса
    "sudo /sbin/service #@@service restart >>/dev/null 2>&1 && \
    sudo /sbin/service #@@service status >>/dev/null 2>&1"
  end
end

module Postfixdomain
  # домен smtp сервиса Postfix
  include Servicedomain
  @@service="postfix"
end
```

Листинг 3. Определение модулей домена Postfix.

Использование полученного модуля Postfixdomain включим в класс Postfixparams, указав директиву «include Postfixdomain».

Точно таким же образом в класс Postfixparams «подмешивается» модуль, определяющий метод P, выполняющий компиляцию кода для функции P(s). Для простоты возьмем за основу формулу Ф-35, так как в нашем примере будет пока очень маленький домен, состоящий только из одной области. Однако, кодирование формул, построенных, как дизъюнкты неудобно. Воспользуемся законом Де-Моргана ([2], страница 27, формула 55) и преобразуем её в конъюнкт:

$$P^0 = !(C \wedge !(B \wedge F \wedge C \wedge D) \wedge ((R \vee t) \wedge (D \vee t) \wedge X)) \quad (\Phi-38)$$

При сложении всех формул в общую последовательность можно компенсировать лидирующий знак отрицания в Ф-38 — удалим его. Так как последний элемент конъюнкта, будучи вычисленным, неизбежно приведет к завершению исполнения, то и перед ним тоже знак отрицания не имеет смысла. Получится совсем простая формула:

$$P^0 = C \wedge !(B \wedge F \wedge C \wedge D) \wedge ((R \vee t) \wedge (D \vee t) \wedge X) \quad (\Phi-39)$$

Вот её-то и положим в основу программной реализации. Кроме этого, в модуль Formulas включим так же метод X, соответствующий функции X(s), который будет выполнять совершенно очевидные действия — печать в поток STDERR сообщения об ошибке и завершение выполнения с кодом «-1» (Листинг 4).

```
module Formulas
```

```

@comment=""
def X # прерывание исполнения
  "{ echo ошибка во время настройки области #@comment >&2 ; exit -1 ; }"
end
def P # выполнение одиночной модификации
#
#! C & !(B & F & C & D) & ((R v I) & (D v I) & X)
#
#! C && {
# !(B && F && C && D) && {
# R ; D ; X
# }
#}
#
"! " + self.C + " && {\n" +
"\t! ( " + self.B + " && ||\n" +
"\t  " + self.F + " && ||\n" +
"\t  " + self.C + " && ||\n" +
"\t  " + self.D + ") && {\n" +
"\t\t" + self.R + "\n" +
"\t\t" + self.D + "\n" +
"\t\t" + self.X + "\n" +
"\t\t}\n" +
"\t;\n"
end
end

```

Листинг 4. Определение модуля Formulas.

Теперь вся необходимая настройка для приведения предметной информационной системы в соответствие с запрограммированной моделью, состоящей только из одной области (Таблица 3), получается путем написания только одной строки кода:

```
Postfixparams.new("myhostname", "mail.example.com", "FQDN почтовой системы")
```

Полный текст примера содержится в Приложении 1. А вот что создает на выходе эта простенькая программка (Листинг 5):

```

! sudo /usr/sbin/postconf myhostname | awk '{print $3}' | grep mail.example.com >>/dev/null 2>&1 && {
  ! ( sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig && \
    sudo /usr/sbin/postconf -e myhostname=mail.example.com && \
    sudo /usr/sbin/postconf myhostname | awk '{print $3}' | grep mail.example.com >>/dev/null 2>&1 && \
    sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1 ) && {
    sudo test -r /etc/postfix/main.cf.orig && sudo /bin/cp /etc/postfix/main.cf.orig /etc/postfix/main.cf
    sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1
    { echo ошибка во время настройки области FQDN почтовой системы >&2 ; exit -1 ; }
  }
}

```

Листинг 5. Код настройки параметра postfix.

Это всего лишь демонстрационный пример, но даже сейчас уже видно, что после задания соответствующей среды, собственно процесс программирования задач системного

администрирования становится совсем не трудоемким.

Практический пример настройки smtp сервера postfix.

Теперь попробуем более серьезный пример. Возьмем задачу, которая уже обсуждалась в части 3 [1]. Тогда был получен весьма несложный листинг программки ([1], часть 3, Листинг 1) автоматической настройки smtp сервера postfix в среде CentOS так, как это предлагалось сделать в официальном HowTo на сайте [4]. То есть, возьмем за основу листинг из приложения 1 и допишем программу, которая автоматически создаст текст решения, пригодного для локального исполнения на настраиваемой системе.

Создаваемая программа, в отличие от первого примера, будет состоять из последовательности множества шагов. Каждый шаг будет обрабатывать одну область. Каждая область, как ранее было замечено, должна принадлежать некоторому домену. Естественно, что области настроек, содержащихся в main.cf, относятся к домену Postfix. А вот, те области, которые определяют не сервис, а условия применения всего алгоритма, можно объединить в абстрактный технический домен Nothing. В программе шаги группируются по принадлежности к доменам и тем самым образуют процедуры. Вот, например, как можно представить последовательность шагов, согласно формуле Ф-39, составляющих некую процедуру (Ф-40), в ходе которой последовательно обрабатываются области $\{s_1, s_2, \dots, s_n\}$ принадлежащие общему домену:

$$\begin{aligned} & !C(s_1) \wedge !(B(s_1) \wedge F(s_1) \wedge C(s_1) \wedge D(s_1)) \wedge ((R(s_1) \vee t) \wedge (D(s_1) \vee t) \wedge X(s_1)) \\ & !C(s_2) \wedge !(B(s_2) \wedge F(s_2) \wedge C(s_2) \wedge D(s_2)) \wedge ((R(s_2) \vee t) \wedge (D(s_2) \vee t) \wedge X(s_2)) \\ & \dots \\ & !C(s_n) \wedge !(B(s_n) \wedge F(s_n) \wedge C(s_n) \wedge D(s_n)) \wedge ((R(s_n) \vee t) \wedge (D(s_n) \vee t) \wedge X(s_n)) \end{aligned}$$

(Ф-40)

Согласно логике построения формулы Ф-39, третий член конъюнкта приводит к прерыванию выполнения не только шага, но и всей процедуры и вообще всего решения. Предположим, что допустимо временно проигнорировать статус возвращаемый $D(s)$. Другими словами, представим, что нас интересует лишь факт вызова функции вычисления домена, например, для рестарта сервиса после настройки его параметров и не более (восстановление опустим)! Тогда можно все вызовы $D(s)$ из второго члена конъюнкта каждого шага вынести за скобки. Например так (формула Ф-41):

$$\begin{aligned} & (!C(s_1) \wedge !(B(s_1) \wedge F(s_1) \wedge C(s_1)) \wedge ((R(s_1) \vee t) \wedge (D(s_1) \vee t) \wedge X(s_1))) \wedge (D(s_1) \vee X(s_1)) \\ & (!C(s_2) \wedge !(B(s_2) \wedge F(s_2) \wedge C(s_2)) \wedge ((R(s_2) \vee t) \wedge (D(s_2) \vee t) \wedge X(s_2))) \wedge (D(s_2) \vee X(s_2)) \\ & \dots \\ & (!C(s_n) \wedge !(B(s_n) \wedge F(s_n) \wedge C(s_n)) \wedge ((R(s_n) \vee t) \wedge (D(s_n) \vee t) \wedge X(s_n))) \wedge (D(s_n) \vee X(s_n)) \end{aligned}$$

(Ф-41)

Согласно рассуждения при выводе формулы Ф-37 все последовательные шаги допустимо рассматривать, как формулы объединенные общей конъюнкцией, и, значит, можно все вызовы функции $D(s)$ сгруппировать в самом конце последовательности шагов (формула Ф-42):

$$\begin{aligned} & !C(s_1) \wedge !(B(s_1) \wedge F(s_1) \wedge C(s_1)) \wedge ((R(s_1) \vee t) \wedge (D(s_1) \vee t) \wedge X(s_1)) \\ & !C(s_2) \wedge !(B(s_2) \wedge F(s_2) \wedge C(s_2)) \wedge ((R(s_2) \vee t) \wedge (D(s_2) \vee t) \wedge X(s_2)) \\ & \dots \end{aligned}$$

$$(!C(s_n) \wedge !(B(s_n) \wedge F(s_n) \wedge C(s_n)) \wedge ((R(s_n) \vee t) \wedge (D(s_n) \vee t) \wedge X(s_n))) \wedge (D(s_1) \wedge \dots \wedge D(s_n) \vee X(s_n))$$

(Ф-42)

Но ведь все вызовы $\mathbf{D}(s_i)$ для s_i принадлежащих одному и тому же домену идентичны (формула Ф-43):

$$D(s_1) \equiv D(s_2) \equiv \dots \equiv D(s_n) \quad (\text{Ф-43})$$

Это дает право заменить их только одним вызовом $\mathbf{D}(s)$ (формула Ф-44). Причем, не важно с каким именно аргументом из указанного домена. Получается примерно тоже самое, что было несколько иным путем построено в ходе обсуждения формул Ф-35 и Ф-37

$$\begin{aligned} & !C(s_1) \wedge !(B(s_1) \wedge F(s_1) \wedge C(s_1)) \wedge ((R(s_1) \vee t) \wedge (D(s_1) \vee t) \wedge X(s_1)) \\ & !C(s_2) \wedge !(B(s_2) \wedge F(s_2) \wedge C(s_2)) \wedge ((R(s_2) \vee t) \wedge (D(s_2) \vee t) \wedge X(s_2)) \\ & \dots \\ & (!C(s_n) \wedge !(B(s_n) \wedge F(s_n) \wedge C(s_n)) \wedge ((R(s_n) \vee t) \wedge (D(s_n) \vee t) \wedge X(s_n))) \wedge (D(s_n) \vee X(s_n)) \end{aligned}$$

(Ф-44)

А теперь представим последовательность формул Ф-44, составляющих некоторое решение. Основным элементом в них будут формулы вида Ф-45:

$$P = !C \wedge !(B \wedge F \wedge C) \wedge ((R \vee t) \wedge (D \vee t) \wedge X) \quad (\text{Ф-45})$$

И только в тех местах последовательности шагов, где один домен будет сменяться другим, к формуле Ф-45 надо будет «подмешивать» дизъюнкт, вычисляющий домен или завершающий последовательность настройки аварийно в случае ошибки вычисления домена. С точки зрения написания программы, что «завершение обработки домена», что «начало обработки следующего домена» все равно. Именно по такому принципу строится основной модуль расчета формул (листинг б) в программе решения настройки smtp сервера (Приложение 2). Программный код «фиксирует» факт смены домена в последовательности генерации формул и в начале следующей процедуры обработки нового домена создает код для завершения обработки сменившегося домена — пересчет его значения или аварийное завершение.

module Formulas

```
# базовый модуль формул
@comment=""
def X (domain="") # прерывание исполнения
  cause="во время"
  if domain != ""
    cause="вычисления домена " + $domainname + " после"
  end
  "{ echo ошибка " + cause + " настройки области #\$stepcomment >&2 ; exit -1 ; }"
end
end
```

module FormulaP

```
include Formulas

def P1 # заключительный пересчет предыдущего домена
```

```

outstr=""
if $domainname != self.DjmainName
  # домен сменился
  if $domainname != "" and $domain != "true"
    # домен был действительным - пересчитаем его
    ostr=
"[ $(cat $fixdone) -ne 0 ] && \\n" +
"\t( " + $domain + " && \\n" +
"\techo успешно вычислен домен " + $domainname + ") || \\n" +
"\t" + self.X("yes") + ")\\n" +
"echo 0 >$fixdone\\n"
    end
    $domain=self.D
    $domainname=self.DomainName
  end
  ostr
end

def P2 # выполнение одиночной модификации
  if $stepcomment != @comment
    # поправим название области
    $stepcomment = @comment
  end
  "echo пропущено >$fixflag\\n" +
"! " + self.C + " && {\\n" +
"\t! ( " + self.B + " && \\n" +
"\t  " + self.F + " && \\n" +
"\t  " + self.C + " && \\n" +
"\t  echo $(($(cat $fixdone)+1)) >$fixdone && \\n" +
"\t  echo выполнено >$fixflag ) && {\\n" +
"\t\t" + self.R + "\\n" +
"\t\t" + self.D + "\\n" +
"\t\t" + self.X + "\\n" +
"\t\t}\\n" +
"\t}\\n" +
"echo $(cat ${fixflag}): #${stepcomment}\\n"
  end
end

```

Листинг 6. Модули расчета формул.

Для передачи параметров «сменившегося» домена используются глобальные переменные с мнемоническими названиями, которые определяются и ликвидируются в специальных вспомогательных классах (листинг 7).

```

class SolutionStart < Nowhereloc
  # техническая абстрактная область
  include Nothingdomain
  include FormulaP
  def initialize (comment0="решение")

```

```

$solutioncomment=comment0
print "#!/bin/bash\n"
print "myhome=$(pwd)\n"
print "myname=${0##*/}\n"
print "fixdone=/tmp/$myname.$$fixdone ; echo 0 >$fixdone\n"
print "fixflag=/tmp/$myname.$$fixflag ; echo >$fixflag\n"
print "cleartmp() { rm -f /tmp/$myname.$$.* ; }\n"
print "trap cleartmp INT TERM EXIT\n"
print "echo #solutioncomment\n"
$domain=""
$stepcomment=""
$domainname=""
end
end

class SolutionDone < Nowhereloc
# техническая абстрактная область
include Nothingdomain
include FormulaP
def initialize ()
print self.P1
print "cleartmp ; trap - INT TERM EXIT\n"
print "echo завершено успешно: #solutioncomment\n"
print "exit 0\n"
end
end

```

Листинг 7. Вспомогательные классы.

В остальном программа из приложения 2 повторяет решения из приложения 1. Завершается листинг из приложения 2 собственно текстом решения по настройке smtp сервера. Решение очень короткое (листинг 8). Начало и конец — вспомогательные области, инициализирующие глобальные переменные и создающие заголовки получаемого кода.

```

SolutionStart.new("Настройка smtp сервера postfix")
Checkos.new("/etc/redhat-release", "CentOS", "проверка платформы")
Checkrpm.new("postfix", "установка пакета")
Postfixparams.new("myhostname", "mail.example.com", "FQDN почтовой системы")
Postfixparams.new("mydomain", "example.com", "почтовый домен")
Postfixparams.new("myorigin", "|||$mydomain", "исходный домен почтовой системы")
Postfixparams.new("inet_interfaces", "all", "прослушиваемые интерфейсы")
Postfixparams.new("mydestination", "|||$myhostname,localhost.|||$mydomain,localhost,|||$mydomain", "принимаемые домены")
Postfixparams.new("mynetworks", "192.168.0.0/24,127.0.0.1/8", "доверенные сети")
Postfixemptyparams.new("relay_domains", "почтовый реллей")
Postfixparams.new("home_mailbox", "Maildir/", "формат почтового ящика")
Setautorun.new("postfix", "автоматический запуск")
SolutionDone.new

```

Листинг 8. Решение по настройке smtp сервера postfix.

Затем последовательно выполняются три процедуры: проверка условий выполнения, настройка postfix, включение автозапуска. Можно область «автоматический запуск» включить в домен Postfix, тогда процедур будет две. Но разумно сначала правильно настроить smtp сервер, так чтобы домен Postfix вычислялся, а уж только потом включать его автозапуск.

Тут уже видно, что в значительной степени разбиение на домены и последовательность действий являются субъективно зависимыми. То есть, ранее декларированная, согласно [2, страница 210, принцип А], счетность и упорядоченность элементов доменов и элементов модели пока никак не ограничивается формально. И это можно продемонстрировать.

Программа из приложения 2 в результате своего выполнения создает shell-код (Приложение 3), который выполняется со следующим результатом (Листинг 9):

```
[root@localhost ~]# ./postfix.rb.sh
Настройка smtp сервера postfix
пропущено: проверка платформы
пропущено: установка пакета
выполнено: FQDN почтовой системы
пропущено: почтовый домен
выполнено: исходный домен почтовой системы
выполнено: прослушиваемые интерфейсы
выполнено: принимаемые домены
выполнено: доверенные сети
выполнено: почтовый реллей
выполнено: формат почтового ящика
успешно вычислен домен Postfix
пропущено: автоматический запуск
завершено успешно: Настройка smtp сервера postfix
```

Листинг 9. Результат выполнения кода из приложения 3.

На листинге 9 видно, что настройка области «почтовый домен» пропускается. Как интересно! Программа построена в точности, как предписано официальной документацией [4], однако, её исполнение не приводит к адекватному эффекту. В main.cf не создается нужный код (Листинг 10):

```
myhostname = mail.example.com
mydomain = example.com
myorigin = $mydomain
inet_interfaces = all
mydestination = $myhostname, localhost.$mydomain, localhost, $mydomain
mynetworks = 192.168.0.0/24, 127.0.0.0/8
relay_domains =
home_mailbox = Maildir/
```

Листинг 10. Код main.cf согласно документации [4].

Область «почтовый домен» совершенно не очевидным образом зависит от области «FQDN почтовой системы». Попробуем поменять очередность их обработки (Листинг 11):

```
SolutionStart.new("Настройка smtp сервера postfix")
```

```

Checkos.new("/etc/redhat-release","CentOS","проверка платформы")
Checkrpm.new("postfix","установка пакета")
Postfixparams.new("mydomain","example.com","почтовый домен")
Postfixparams.new("myhostname","mail.example.com","FQDN почтовой системы")
Postfixparams.new("myorigin","\\$mydomain","исходный домен почтовой системы")
Postfixparams.new("inet_interfaces","all","прослушиваемые интерфейсы")
Postfixparams.new("mydestination","\\$myhostname,localhost.\\$mydomain,localhost,\\$mydomain","принимаемые домены")
Postfixparams.new("mynetworks","192.168.0.0/24,127.0.0.1/8","доверенные сети")
Postfixemptyparams.new("relay_domains","почтовый реллей")
Postfixparams.new("home_mailbox","Maildir/","формат почтового ящика")
Setautorun.new("postfix","автоматический запуск")
SolutionDone.new

```

Листинг 11. Модифицированное решение по настройке smtp сервера postfix.

После прогона полученного кода результат будет совершенно другим (Листинг 12):

```

[root@localhost ~]# ./postfix.rb.sh.new
Настройка smtp сервера postfix
пропущено: проверка платформы
пропущено: установка пакета
выполнено: почтовый домен
выполнено: FQDN почтовой системы
выполнено: исходный домен почтовой системы
выполнено: прослушиваемые интерфейсы
выполнено: принимаемые домены
выполнено: доверенные сети
выполнено: почтовый реллей
выполнено: формат почтового ящика
успешно вычислен домен Postfix
пропущено: автоматический запуск
завершено успешно: Настройка smtp сервера postfix

```

Листинг 12. Результат исполнения модифицированного кода из приложения 3.

Все области были обработаны и настроены как следует и в файле main.cf в этом случае созданы все необходимые настройки (Листинг 13):

```

[root@localhost postfix]# diff -Naur main.cf.old main.cf.new
--- main.cf.old 2011-08-25 01:03:34.760777034 +0400
+++ main.cf.new 2011-08-25 01:03:11.869827691 +0400
@@ -674,6 +674,7 @@
# readme_directory: The location of the Postfix README files.
#
readme_directory = /usr/share/doc/postfix-2.6.6/README_FILES
+mydomain = example.com
myhostname = mail.example.com
myorigin = $mydomain
mynetworks = 192.168.0.0/24,127.0.0.1/8

```

Листинг 13. Проверка результата исполнения модифицированного скрипта.

Хочется обратить особое внимание, что в описании настройки в HowTo на сайте [4] данная особенность предметной области никак не проявляется и не отмечается. То есть формальный подход к решению задачи позволил точнее описать и задачу и её решение.

Приведенный в этой статье и в приложениях к ней код Ruby не может служить образцом мастерства программирования. Он создавался не программистом, а все лишь системным администратором для демонстрации формального подхода к решению задач системного администрирования. Тем не менее данный пример полностью решает поставленную задачу и демонстрирует действенность предложенного подхода к автоматизации системного администрирования.

Предварительные выводы.

Хотя, задача и решена, но остаются еще вопросы об удобстве поддержки и модификации созданного решения. Листинг 11 не представляется достаточно наглядным. В нем много условностей, привнесенных синтаксическими особенностями платформы реализации. Но можно попробовать снова все вернуть в табличную форму. Вот как будет выглядеть созданное решение в виде таблицы (Таблица 4):

Домен	Класс области	Область <i>s</i>	Область <i>z</i>	Значение <i>v</i>
(1)	(2)	(3)	(4)	(5)
<i>Nothing</i>	<i>Checkos</i>	<i>проверка платформы</i>	<i>/etc/redhat-release</i>	<i>CentOS</i>
<i>Nothing</i>	<i>Checkrpm</i>	<i>установка пакета</i>	<i>Набор установленных пакетов</i>	<i>postfix</i>
<i>Postfix</i>	<i>Postfixparams</i>	<i>почтовый домен</i>	<i>/etc/postfix/main.cf, mydomain</i>	<i>example.com</i>
<i>Postfix</i>	<i>Postfixparams</i>	<i>FQDN почтовой системы</i>	<i>/etc/postfix/main.cf, myhostname</i>	<i>mail.example.com</i>
<i>Postfix</i>	<i>Postfixparams</i>	<i>исходный домен почтовой системы</i>	<i>/etc/postfix/main.cf, myorigin</i>	<i>\$mydomain</i>
<i>Postfix</i>	<i>Postfixparams</i>	<i>прослушиваемые интерфейсы</i>	<i>/etc/postfix/main.cf, inet_interfaces</i>	<i>all</i>
<i>Postfix</i>	<i>Postfixparams</i>	<i>принимаемые домены</i>	<i>/etc/postfix/main.cf, mydestination</i>	<i>\$myhostname, localhost, \$mydomain, localhost, \$mydomain</i>
<i>Postfix</i>	<i>Postfixparams</i>	<i>доверенные сети</i>	<i>/etc/postfix/main.cf, mynetworks</i>	<i>192.168.0.0/24, 127.0.0.1/8</i>
<i>Postfix</i>	<i>Postfixemptyparams</i>	<i>почтовый реллей</i>	<i>/etc/postfix/main.cf, relay_domains</i>	

<i>Postfix</i>	<i>Postfixparams</i>	<i>формат почтового ящика</i>	<i>/etc/postfix/main.cf, home_mailbox</i>	<i>Maildir/</i>
<i>Nothing</i>	<i>Setautorun</i>	<i>автоматический запуск</i>	<i>Скрипты автозапуска</i>	<i>postfix</i>

Таблица 4. Табличное описание настройки smtp сервера postfix.

В каждой строке таблицы 4 описан один шаг. Шаги отсортированы по первой колонке, в которой указан домен. В колонке 3 название области в модели **S**. В колонке 4 эквивалентное условное наименование в терминах реальной системы **Z**. В колонке 5 значение, присвоенное области в модели **S**. В таком виде информация по решению вполне наглядна и даже читается со смыслом. Однако, содержимое колонки 4 не соответствует формату вызова методов из листинга 11. В колонке 4 перечислены все необходимые параметры для получения значения области **z** реальной системы. В листинге 11 в процессе наследования свойств часть параметров «скрылась под слоем» инкапсуляции. Здесь же пришлось добавить дополнительные значения для точного описания алгоритма. Если задача еще более усложниться, то содержимое колонки 4 потребуются расширить так, чтобы разместить там все значения квалифицирующие обрабатываемую область системы **Z** в полной мере.

В приложении 2 представлен пример решения тривиальной задачи. В реальной ситуации текст генератора скриптов, скорее всего, будет несколько более сложным. Для создания практически ценных решений был создан проект YAMIS [5], размещенный в Сети на условиях открытой лицензии GPL, так чтобы им мог воспользоваться каждый. Там не много готовых скриптов, но список имеет шансы на пополнение в процессе дальнейшей работы над темой автоматизации системного программирования.

Одна из главных причин возникновения отличий практических текстов [5] от демонстрационных в декларированном выше стремлении достичь рабочего компромисса между удобством, сопровождающим инкапсуляцию методов в ООП, и информативностью конечного представления данных о решении. Например с помощью увеличения числа аргументов, используемых в вызовах методов так, чтобы вся необходимая для однозначного описания обрабатываемых областей информация по-возможности была представлена в тексте решения. Объясним почему так происходит.

Фактически описание области задается в виде списка аргументов метода *initialize*. Домен присутствует в неявном виде. Значение *s* соответствует аргументу *comment*, дающему области название удобное для восприятия человека, которое используется в модели системы **S**. Оставшиеся параметры разделяются на *param** и *value**. *Param** задают указание на область **z** реальной системы **Z**. Именно *param** должен указывать на файл, если область имеет файловую форму хранения. *Value** описывают значения, которые используются при детектировании (выборке) и изменении состояния области. Когда потребуется более одного значения *value**? Тогда, когда для детектирования и изменения нужно будет задавать разные строки. Например, если для правильной настройки области потребуется выбрать одну из строк группы с подобным форматом и изменить только её, тогда образец для поиска может не совпадать с образцом замены. А когда нужно более одного значения *param**? Тогда, когда кроме пути и названия файла потребуется как-то квалифицировать информационную группу внутри самого файла. Например, указать секцию настройки. Или иной пример. Допустим, кроме пути и имени обрабатываемого файла нужно будет указать еще и архив с источником исходных данных для выполнения преобразования (установка пакета, компиляция ПО из исходных текстов, и так далее...).

На самом деле, рассмотрение всех возможных форм и видов областей информационной системы и способов представления их в модели является предметом особого разговора,

который скорее всего не имеет шансов на конечное завершение, поскольку он может закончиться только с познанием всех возможных способов представления информации в вычислительных системах, которые в свою очередь тоже развиваются и тем самым множат упомянутые способы. Потому, в примерах, приведенных в статье, было достаточно лишь самых простых способов описания областей. А вот, в текстах проекта [5], размещенного в Интернете, используются уже более изощренные форматы задания областей.

Ну и, наконец, в реальной модели придется учитывать те факты, что одна область может принадлежать не только одному домену, и то, что при ошибке на некотором шаге восстановлению подлежит не только последняя измененная область, а все области, подвергшиеся модификации в ходе исполнения всех шагов процедуры, включая и последний, если нет гарантии выполнения всей совокупности настроек в соответствие с моделью S одновременно.

Другим недостатком столь прямолинейного подхода к инструментальному решению является невозможность преобразования полученных результатов в форму иную кроме той, что соответствует привлеченным средствам. В данном случае, языку Ruby. Если бы в качестве средства решения задачи был бы выбран Perl, то в итоге была бы получена программа на языке Perl. Но ведь в заявленной цели стоит не программирование на каком-то из языков, а решение задач администрирования информационных систем. Даже более, как было еще раз декларировано в самом начале данной статьи: накопление опыта администрирования! Какое же тут происходит накопление, если в ходе решения часть информации теряется? Кроме того, если задаться вопросом, что в большей мере соответствует модели информационной системы листинг 11 или таблица 4, то надо признаться, что скорее это будет таблица 4. Именно на основании таблицы 4 можно построить иной алгоритм, приводящий к эквивалентному решению задачи. А вот, листинг 11 является конечной формой, пригодной только для подачи Ruby-процессору. Эти и другие недостатки и методы их преодоления послужат предметом обсуждения в продолжении работы над темой статьи в следующих частях этой публикации.

31.08.2011

Литература и ссылки.

1.Предыдущие части цикла по автоматизации системного администрирования.
<http://www.barabanov.ru/arts.html>

2. С. К. Клини. Математическая логика. Изд.4-е. 2008г.

3. Майкл Фитцджеральд. Изучаем Ruby. Пер. С англ. 2008г.

4.Руководство по настройке почтовой системы CentOS. Postfix HOWTO.
<http://wiki.centos.org/HowTos/postfix>

5.Проект YAMIS по разработке системы автоматизации администрирования.
<http://yamis.sourceforge.net>

Приложение 1.

Полный текст примера реализации одного шага настройки.

```

#!/usr/bin/env ruby
#

class Fileloc
  # область представленная файлом
  @@filename=""
  def B # резервное копирование
    "sudo cp #@@filename #@@filename.orig"
  end
  def R # восстановление
    "sudo test -r #@@filename.orig && sudo /bin/cp #@@filename.orig #@@filename"
  end
end

module Servicedomain
  # домен, определяемый отношением к сервисному процессу
  @@service=""
  def D # проверка вычислимости домена-сервиса
    "sudo /sbin/service #@@service restart >>/dev/null 2>&1 &&" +
    "sudo /sbin/service #@@service status >>/dev/null 2>&1"
  end
end

module Postfixdomain
  # домен smtp сервиса Postfix
  include Servicedomain
  @@service="postfix"
end

module Formulas
  @comment=""
  attr_accessor :comment
  def X # прерывание исполнения
    "{ echo ошибка во время настройки области #@comment >&2 ; exit -1 ; }"
  end
  def P # выполнение одиночной модификации
  #
  #! C & !(B & F & C & D) & ((R v I) & (D v I) & X)
  #
  #! C && {
  # !(B && F && C && D) && {
  # R ; D ; X
  # }
  #}
  #
  "! " + self.C + " && {\n" +
  "\t! ( " + self.B + " && ||\n" +
  "\t " + self.F + " && ||\n" +
  "\t " + self.C + " && ||\n" +
  "\t " + self.D + ") && {\n" +

```

```

"\t\t" + self.R + "\n" +
"\t\t" + self.D + "\n" +
"\t\t" + self.X + "\n" +
"\t\t}\n" +
"\t}\n"
end
end

class Postfixparams < Fileloc
  # область представленная параметрической строкой в формате postfix main.cf
  include Postfixdomain
  include Formulas
  def initialize (param0,value0,comment0="")
    @@filename="/etc/postfix/main.cf"
    @comment=comment0
    @param=param0
    @value=value0
    print self.P
  end
  def C # проверка
    "sudo /usr/sbin/postconf #@param | awk '{print $3}' | grep #@value >>/dev/null 2>&1"
  end
  def F # модификация
    "sudo /usr/sbin/postconf -e #@param=#@value"
  end
end

Postfixparams.new("myhostname","mail.example.com","FQDN почтовой системы")

```

Приложение 2.

Полный текст примера настройки smtp сервера postfix.

```

#!/usr/bin/env ruby
#

##### базовые классы
class Nowhereloc
  # область не имеющая представления
  def B # резервное копирование
    "true"
  end
  def R # восстановление
    "true"
  end
end

class Fileloc
  # область представленная файлом

```

```

@@filename=""
def B # резервное копирование
  "sudo test -r #@@filename.orig || sudo cp #@@filename #@@filename.orig"
end
def R # восстановление
  "sudo test -r #@@filename.orig && sudo /bin/cp #@@filename.orig #@@filename"
end
end

```

```
##### домены
```

```

module Nothingdomain
  # домен независимых настроек
  def D
    "true"
  end
  def DomainName
    "Nothing"
  end
end

```

```

module Servicedomain
  # домен, определяемый отношением к сервисному процессу
  @@service=""
  def D # проверка вычислимости домена-сервиса
    "sudo /sbin/service #@@service restart >>/dev/null 2>&1 &&" +
    "sudo /sbin/service #@@service status >>/dev/null 2>&1"
  end
end

```

```

module Postfixdomain
  # домен smtp сервиса Postfix
  include Servicedomain
  @@service="postfix"
  def DomainName
    "Postfix"
  end
end

```

```
##### формулы
```

```

#
# для всех областей домена кроме последней
#! C & !(B & F & C) & ((R v I) & (D v I) & X)
#
#! C && {
# !(B && F && C) && {
# R ; D ; X
# }
#}
#
# для последней области

```

```

#(! C & !(B & F & C) & ((R v I) & (D v I) & X)) & (D v X)
#
#! C && {
# !(B && F && C) && {
# R ; D ; X
# }
#}
#D || X
#
#

```

module Formulas

```

# базовый модуль формул
@comment=""
def X (domain="") # прерывание исполнения
  cause="во время"
  if domain != ""
    cause="вычисления домена " + $domainname + " после"
  end
  "{ echo ошибка " + cause + " настройки области # $stepcomment >&2 ; exit -1 ; }"
end
end

```

module FormulaP

include Formulas

```

def P1 # заключительный пересчет предыдущего домена
  outstr=""
  if $domainname != self.DomainName
    # домен сменился
    if $domainname != "" and $domain != "true"
      # домен был действительным - пересчитаем его
      outstr=
"[ $(cat $fixdone) -ne 0 ] && \\n" +
"\t( ( " + $domain + " && \\n" +
"\techo успешно вычислен домен " + $domainname + " ) || \\n" +
"\t" + self.X("yes") + " )\n" +
"echo 0 > $fixdone\n"
    end
    $domain=self.D
    $domainname=self.DomainName
  end
  outstr
end

```

def P2 # выполнение одиночной модификации

```

if $stepcomment != @comment
  # поправим название области
  $stepcomment = @comment
end

```

```

"echo пропущено >$fixflag\n" +
"! " + self.C + " && {\n" +
"\t! ( " + self.B + " && ||\n" +
"\t  " + self.F + " && ||\n" +
"\t  " + self.C + " && ||\n" +
"\t  echo $(($(cat $fixdone)+1)) >$fixdone && ||\n" +
"\t  echo выполнено >$fixflag ) && {\n" +
"\tt" + self.R + "\n" +
"\tt" + self.D + "\n" +
"\tt" + self.X + "\n" +
"\tt}\n" +
"\t}\n" +
"echo $(cat ${fixflag}): #stepcomment\n"
end
end

##### классы областей
class SolutionStart < Nowhereloc
# техническая абстрактная область
include Nothingdomain
include FormulaP
def initialize (comment0="решение")
$solutioncomment=comment0
print "#!/bin/bash\n"
print "myhome=$(pwd)\n"
print "myname=${0##*/}\n"
print "fixdone=/tmp/$myname.$$fixdone ; echo 0 >$fixdone\n"
print "fixflag=/tmp/$myname.$$fixflag ; echo >$fixflag\n"
print "cleartmp() { rm -f /tmp/$myname.$$.* ; }\n"
print "trap cleartmp INT TERM EXIT\n"
print "echo #solutioncomment\n"
$domain=""
$stepcomment=""
$domainname=""
end
end

class SolutionDone < Nowhereloc
# техническая абстрактная область
include Nothingdomain
include FormulaP
def initialize ()
print self.P1
print "cleartmp ; trap - INT TERM EXIT\n"
print "echo завершено успешно: #solutioncomment\n"
print "exit 0\n"
end
end

class Checkos < Nowhereloc

```

```

# область представленная значением, хранимым в специальном файле
include Nothingdomain
include FormulaP
def initialize (param0,value0,comment0="")
  @comment=comment0
  @param=param0
  @value=value0
  print self.P1
  print self.P2
end
def C # проверка
  "grep #@value #@param >>/dev/null 2>&1"
end
def F # модификация
  "false"
end
end

class Checkrpm < Nowhereloc
# область представленная установкой пакета
include Nothingdomain
include FormulaP
def initialize (value0,comment0="")
  @comment=comment0
  @value=value0
  print self.P1
  print self.P2
end
def C # проверка
  "rpm -q #@value >>/dev/null 2>&1"
end
def F # модификация
  "sudo yum install -y #@value"
end
end

class Postfixparams < Fileloc
# область представленная параметрической строкой в формате postfix main.cf
include Postfixdomain
include FormulaP
def initialize (param0,value0,comment0="")
  @@filename="/etc/postfix/main.cf"
  @comment=comment0
  @param=param0
  @value=value0
  print self.P1
  print self.P2
end
def C # проверка
  "sudo /usr/sbin/postconf #@param | awk '{print $3}' | grep \"#@value\" >>/dev/null 2>&1"
end
end

```

```

end
def F # модификация
  "sudo /usr/sbin/postconf -e #{@param}=#@value"
end
end

```

```

class Postfixemptyparams < Fileloc
  # область представленная параметрической строкой в формате postfix main.cf
  include Postfixdomain
  include FormulaP
  def initialize (param0,comment0="")
    @@filename="/etc/postfix/main.cf"
    @comment=comment0
    @param=param0
    print self.P1
    print self.P2
  end
  def C # проверка
    "sudo /usr/sbin/postconf #{@param} | awk '{print $3}' | grep |" ^$ | " >>/dev/null 2>&1"
  end
  def F # модификация
    "sudo /usr/sbin/postconf -e #{@param}=#@"
  end
end
end

```

```

class Setautorun < Nowhereloc
  # область представленная настройкой автоматического запуска
  include Nothingdomain
  include FormulaP
  def initialize (value0,comment0="")
    @comment=comment0
    @value=value0
    print self.P1
    print self.P2
  end
  def C # проверка
    "LC_ALL=C sudo /sbin/chkconfig --list #{@value} | grep $(runlevel | awk '{print $2}'):on"
    >>/dev/null 2>&1"
  end
  def F # модификация
    "sudo /sbin/chkconfig #{@value} on"
  end
end
end

```

```

##### собственно настройка
SolutionStart.new("Настройка smtp сервера postfix")
Checkos.new("/etc/redhat-release", "CentOS", "проверка платформы")
Checkrpm.new("postfix", "установка пакета")
Postfixparams.new("myhostname", "mail.example.com", "FQDN почтовой системы")
Postfixparams.new("mydomain", "example.com", "почтовый домен")

```

```

Postfixparams.new("myorigin", "\\$mydomain", "исходный домен почтовой системы")
Postfixparams.new("inet_interfaces", "all", "прослушиваемые интерфейсы")
Postfixparams.new("mydestination", "\\$myhostname,localhost.\\$mydomain,localhost,\\$mydomain", "принимаемые домены")
Postfixparams.new("mynetworks", "192.168.0.0/24,127.0.0.1/8", "доверенные сети")
Postfixemtpyparams.new("relay_domains", "почтовый реллей")
Postfixparams.new("home_mailbox", "Maildir/", "формат почтового ящика")
Setautorun.new("postfix", "автоматический запуск")
SolutionDone.new

```

Приложение 3.

Текст скрипта, созданного генератором из Приложения 2.

```

#!/bin/bash
myhome=$(pwd)
myname=${0##*/}
fixdone=/tmp/$myname.$$fixdone ; echo 0 >$fixdone
fixflag=/tmp/$myname.$$fixflag ; echo >$fixflag
cleartmp() { rm -f /tmp/$myname.$$* ; }
trap cleartmp INT TERM EXIT
echo Настройка smtp сервера postfix
echo пропущено >$fixflag
! grep CentOS /etc/redhat-release >>/dev/null 2>&1 && {
    ! ( true && \
        false && \
        grep CentOS /etc/redhat-release >>/dev/null 2>&1 && \
        echo $((cat $fixdone)+1)) >$fixdone && \
        echo выполнено >$fixflag ) && {
        true
        true
        { echo ошибка во время настройки области проверка платформы >&2 ; exit -1 ; }
    }
}
echo $(cat $fixflag): проверка платформы
echo пропущено >$fixflag
! rpm -q postfix >>/dev/null 2>&1 && {
    ! ( true && \
        sudo yum install -y postfix && \
        rpm -q postfix >>/dev/null 2>&1 && \
        echo $((cat $fixdone)+1)) >$fixdone && \
        echo выполнено >$fixflag ) && {
        true
        true
        { echo ошибка во время настройки области установка пакета >&2 ; exit -1 ; }
    }
}
echo $(cat $fixflag): установка пакета
echo пропущено >$fixflag
! sudo /usr/sbin/postconf myhostname | awk '{print $3}' | grep "mail.example.com" >>/dev/null 2>&1 && {
    ! ( sudo test -r /etc/postfix/main.cf.orig || sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig && \
        sudo /usr/sbin/postconf -e myhostname=mail.example.com && \
        sudo /usr/sbin/postconf myhostname | awk '{print $3}' | grep "mail.example.com" >>/dev/null 2>&1 && \
        echo $((cat $fixdone)+1)) >$fixdone && \
        echo выполнено >$fixflag ) && {
        sudo test -r /etc/postfix/main.cf.orig && sudo /bin/cp /etc/postfix/main.cf.orig /etc/postfix/main.cf
        sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1
        { echo ошибка во время настройки области FQDN почтовой системы >&2 ; exit -1 ; }
    }
}
echo $(cat $fixflag): FQDN почтовой системы
echo пропущено >$fixflag
! sudo /usr/sbin/postconf mydomain | awk '{print $3}' | grep "example.com" >>/dev/null 2>&1 && {

```

```

! ( sudo test -r /etc/postfix/main.cf.orig || sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig && |
  sudo /usr/sbin/postconf -e mydomain=example.com && |
  sudo /usr/sbin/postconf mydomain | awk '{print $3}' | grep "example.com" >>/dev/null 2>&1 && |
  echo $((cat $fixdone)+1)) >$fixdone && |
  echo выполнено >$fixflag ) && {
  sudo test -r /etc/postfix/main.cf.orig && sudo /bin/cp /etc/postfix/main.cf.orig /etc/postfix/main.cf
  sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1
  { echo ошибка во время настройки области почтовый домен >&2 ; exit -1 ; }
}
}
echo $(cat $fixflag): почтовый домен
echo пропущено >$fixflag
! sudo /usr/sbin/postconf myorigin | awk '{print $3}' | grep "\$mydomain" >>/dev/null 2>&1 && {
! ( sudo test -r /etc/postfix/main.cf.orig || sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig && |
  sudo /usr/sbin/postconf -e myorigin=\$mydomain && |
  sudo /usr/sbin/postconf myorigin | awk '{print $3}' | grep "\$mydomain" >>/dev/null 2>&1 && |
  echo $((cat $fixdone)+1)) >$fixdone && |
  echo выполнено >$fixflag ) && {
  sudo test -r /etc/postfix/main.cf.orig && sudo /bin/cp /etc/postfix/main.cf.orig /etc/postfix/main.cf
  sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1
  { echo ошибка во время настройки области исходный домен почтовой системы >&2 ; exit -1 ; }
}
}
echo $(cat $fixflag): исходный домен почтовой системы
echo пропущено >$fixflag
! sudo /usr/sbin/postconf inet_interfaces | awk '{print $3}' | grep "all" >>/dev/null 2>&1 && {
! ( sudo test -r /etc/postfix/main.cf.orig || sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig && |
  sudo /usr/sbin/postconf -e inet_interfaces=all && |
  sudo /usr/sbin/postconf inet_interfaces | awk '{print $3}' | grep "all" >>/dev/null 2>&1 && |
  echo $((cat $fixdone)+1)) >$fixdone && |
  echo выполнено >$fixflag ) && {
  sudo test -r /etc/postfix/main.cf.orig && sudo /bin/cp /etc/postfix/main.cf.orig /etc/postfix/main.cf
  sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1
  { echo ошибка во время настройки области прослушиваемые интерфейсы >&2 ; exit -1 ; }
}
}
echo $(cat $fixflag): прослушиваемые интерфейсы
echo пропущено >$fixflag
! sudo /usr/sbin/postconf mydestination | awk '{print $3}' | grep "\$myhostname,localhost.\$mydomain,localhost,\$mydomain"
>>/dev/null 2>&1 && {
! ( sudo test -r /etc/postfix/main.cf.orig || sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig && |
  sudo /usr/sbin/postconf -e mydestination=\$myhostname,localhost.\$mydomain,localhost,\$mydomain && |
  sudo /usr/sbin/postconf mydestination | awk '{print $3}' | grep "\$myhostname,localhost.\$mydomain,localhost,\$mydomain"
  >>/dev/null 2>&1 && |
  echo $((cat $fixdone)+1)) >$fixdone && |
  echo выполнено >$fixflag ) && {
  sudo test -r /etc/postfix/main.cf.orig && sudo /bin/cp /etc/postfix/main.cf.orig /etc/postfix/main.cf
  sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1
  { echo ошибка во время настройки области принимаемые домены >&2 ; exit -1 ; }
}
}
echo $(cat $fixflag): принимаемые домены
echo пропущено >$fixflag
! sudo /usr/sbin/postconf mynetworks | awk '{print $3}' | grep "192.168.0.0/24,127.0.0.1/8" >>/dev/null 2>&1 && {
! ( sudo test -r /etc/postfix/main.cf.orig || sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig && |
  sudo /usr/sbin/postconf -e mynetworks=192.168.0.0/24,127.0.0.1/8 && |
  sudo /usr/sbin/postconf mynetworks | awk '{print $3}' | grep "192.168.0.0/24,127.0.0.1/8" >>/dev/null 2>&1 && |
  echo $((cat $fixdone)+1)) >$fixdone && |
  echo выполнено >$fixflag ) && {
  sudo test -r /etc/postfix/main.cf.orig && sudo /bin/cp /etc/postfix/main.cf.orig /etc/postfix/main.cf
  sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1
  { echo ошибка во время настройки области доверенные сети >&2 ; exit -1 ; }
}
}
echo $(cat $fixflag): доверенные сети
echo пропущено >$fixflag
! sudo /usr/sbin/postconf relay_domains | awk '{print $3}' | grep "^$" >>/dev/null 2>&1 && {

```

```

! ( sudo test -r /etc/postfix/main.cf.orig || sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig && |
  sudo /usr/sbin/postconf -e relay_domains="" && |
  sudo /usr/sbin/postconf relay_domains | awk '{print $3}' | grep "^$" >>/dev/null 2>&1 && |
  echo $((cat $fixdone)+1)) >$fixdone && |
  echo выполнено >$fixflag ) && {
  sudo test -r /etc/postfix/main.cf.orig && sudo /bin/cp /etc/postfix/main.cf.orig /etc/postfix/main.cf
  sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1
  { echo ошибка во время настройки области почтовый реллей >&2 ; exit -1 ; }
}
}
echo $(cat $fixflag): почтовый реллей
echo пронущено >$fixflag
! sudo /usr/sbin/postconf home_mailbox | awk '{print $3}' | grep "Maildir/" >>/dev/null 2>&1 && {
! ( sudo test -r /etc/postfix/main.cf.orig || sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig && |
  sudo /usr/sbin/postconf -e home_mailbox=Maildir/ && |
  sudo /usr/sbin/postconf home_mailbox | awk '{print $3}' | grep "Maildir/" >>/dev/null 2>&1 && |
  echo $((cat $fixdone)+1)) >$fixdone && |
  echo выполнено >$fixflag ) && {
  sudo test -r /etc/postfix/main.cf.orig && sudo /bin/cp /etc/postfix/main.cf.orig /etc/postfix/main.cf
  sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1
  { echo ошибка во время настройки области формат почтового ящика >&2 ; exit -1 ; }
}
}
echo $(cat $fixflag): формат почтового ящика
[ $(cat $fixdone) -ne 0 ] && \
  ( ( sudo /sbin/service postfix restart >>/dev/null 2>&1 && sudo /sbin/service postfix status >>/dev/null 2>&1 && |
    echo успешно вычислен домен Postfix ) || |
  { echo ошибка вычисления домена Postfix после настройки области формат почтового ящика >&2 ; exit -1 ; } )
echo 0 >$fixdone
echo пронущено >$fixflag
! LC_ALL=C sudo /sbin/chkconfig --list postfix | grep $(runlevel | awk '{print $2}'):on >>/dev/null 2>&1 && {
! ( true && |
  sudo /sbin/chkconfig postfix on && |
  LC_ALL=C sudo /sbin/chkconfig --list postfix | grep $(runlevel | awk '{print $2}'):on >>/dev/null 2>&1 && |
  echo $((cat $fixdone)+1)) >$fixdone && |
  echo выполнено >$fixflag ) && {
  true
  true
  { echo ошибка во время настройки области автоматический запуск >&2 ; exit -1 ; }
}
}
}
echo $(cat $fixflag): автоматический запуск
clearmp ; trap - INT TERM EXIT
echo завершено успешно: Настройка smtp сервера postfix
exit 0

```