РОССИЙСКАЯ ФЕДЕРАЦИЯ

(19) **RU** (11) **2 367 999** (13) **C2**

(51) МПК
G06F 9/44 (2006.01)

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ,
ПАТЕНТАМ И ТОВАРНЫМ ЗНАКАМ

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(21), (22) Заявка: **2005115980/09, 22.07.2004**

(24) Дата начала отсчета срока действия патента:
22.07.2004

(30) Конвенционный приоритет:
24.10.2003 US 10/693,396

(43) Дата публикации заявки: **20.01.2006**

(45) Опубликовано: **20.09.2009** Бюл. № **26**

(56) Список документов, цитированных в отчете о поиске: **US 6208339 B1, 27.03.2001. RU 2187889 C2, 20.08.2002. US 2003/0084197 A1, 01.05.2003.**

(85) Дата перевода заявки PCT на национальную фазу: **25.05.2005**

(86) Заявка PCT:
US 2004/023365 (22.07.2004)

(87) Публикация PCT:
WO 2005/045561 (19.05.2005)

Адрес для переписки:
129090, Москва, ул. Б.Спасская, 25, стр.3, ООО "Юридическая фирма Городисский и Партнеры", пат.пов. Ю.Д.Кузнецову, рег.№ 595

(72) Автор(ы):
СНОУВЕР Джеффри П. (US),
ТРУЕР III Джеймс В. (US),
ПУШПАВАНАМ Каушик (US),
ВУСВАНАТХАН Субраманиан (US)

(73) Патентообладатель(и):
МАЙКРОСОФТ КОРПОРЕЙШН (US)

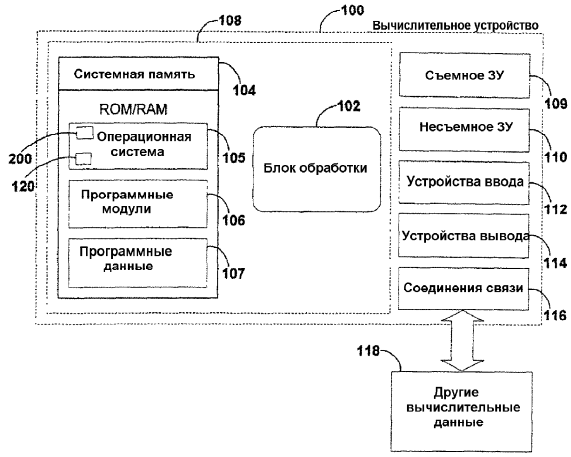
(54) МЕХАНИЗМ ДЛЯ ПОЛУЧЕНИЯ И ПРИМЕНЕНИЯ ОГРАНИЧЕНИЙ К ЛОГИЧЕСКИМ СТРУКТУРАМ В ИНТЕРАКТИВНОЙ СРЕДЕ

(57) Реферат:

Изобретение относится к интерактивным средам, в частности к получению и применению ограничений в интерактивной среде. Техническим результатом является предоставление возможности пользователям определять ограничения интерактивным способом. Предложенный механизм получает ограничения в интерактивной среде, ассоциирует эти ограничения с логическими структурами и затем применяет эти ограничения к логическим структурам при их появлении. Ограничения могут сохраняться в метаданных, связанных с соответствующей логической структурой. Ограничения могут определять тип данных для логической структуры, указатель предиката, указатель документирования, указатель синтаксического анализа, указатель генерации данных, указатель подтверждения данных или указатель обработки и кодирования объектов. Ограничения являются расширяемыми для обеспечения поддержки других указателей. 3 н. и 21 з.п. ф-лы, 23 ил., 8 табл.

RU 2367999 C2

RU 2367999 C2



Фиг.1

RU 2367999 C2

RUSSIAN FEDERATION

FEDERAL SERVICE
FOR INTELLECTUAL PROPERTY,
PATENTS AND TRADEMARKS

(19) **RU** (11) **2 367 999** (13) **C2**

(51) Int. Cl.
G06F 9/44 (2006.01)

(12) ABSTRACT OF INVENTION

(21), (22) Application: **2005115980/09, 22.07.2004**

(24) Effective date for property rights:
22.07.2004

(30) Priority:
24.10.2003 US 10/693,396

(43) Application published: **20.01.2006**

(45) Date of publication: **20.09.2009 Bull. 26**

(85) Commencement of national phase: **25.05.2005**

(86) PCT application:
US 2004/023365 (22.07.2004)

(87) PCT publication:
WO 2005/045561 (19.05.2005)

Mail address:
129090, Moskva, ul. B.Spasskaja, 25, str.3, OOO "Juridicheskaja firma Gorodisskij i Partnery", pat.pov. Ju.D.Kuznetsovu, reg.№ 595

(72) Inventor(s):
SNOUVER Dzheffri P. (US),
TRUER III Dzhejms V. (US),
PUShPAVANAM Kaushik (US),
VUSVANATKhAN Subramanian (US)

(73) Proprietor(s):
MAJKROSOFT KORPOREJShN (US)

RU 2367999 C2

RU 2367999 C2

(54) MECHANISM FOR PRODUCTION AND APPLICATION OF LIMITATIONS TO LOGICAL STRUCTURES IN INTERACTIVE MEDIUM

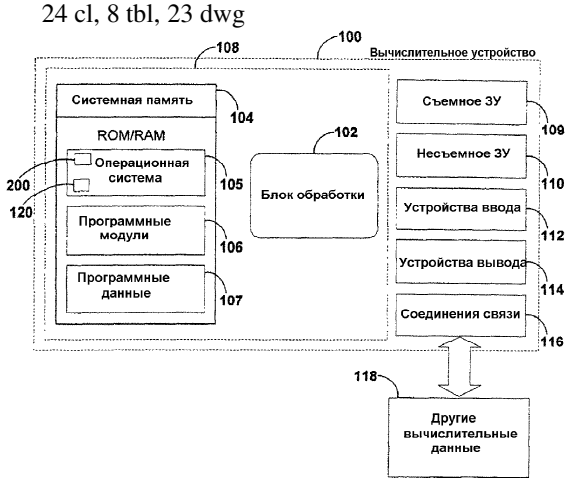
(57) Abstract:

FIELD: physics, computer engineering.

SUBSTANCE: invention is related to interactive mediums, in particular to production and application of limitations in interactive medium. Suggested mechanism receives limitations in interactive medium, associates these limitations with logical structures and then applies these limitations to logical structures in case of their occurrence. Limitations may be preserved in metadata related to corresponding logical structure. Limitations may define type of data for logical structure, predicate indicator, documentation indicator, syntactical analysis indicator, data generation indicator, indicator of data confirmation or indicator of objects processing and coding. Limitations are expandable for provision of support for other indicators.

EFFECT: presentation of possibility to users to define limitations by interactive method.

24 cl, 8 tbl, 23 dwg



Фиг.1

Область техники

Настоящее изобретение относится к интерактивным средам, более конкретно к получению и применению ограничений в интерактивной среде.

Предшествующий уровень техники

В принципе имеется два типа кодов: скомпилированный код и интерпретируемый код. В прошлом скомпилированный код компилировался для получения объектного кода и затем связывался с другими объектными кодами для создания исполняемого кода, который исполнялся во время прогона. В настоящее время в некоторых средах скомпилированный код включает в себя исходный код, который был скомпилирован в промежуточной форме. Во время прогона промежуточная форма компилируется в собственный код для исполнения. В любом из этих сценариев разработчик может определить тип для каждой логической структуры, программируемой в исходном коде. Типы включают в себя целочисленный тип, строковый тип, тип с плавающей точкой и т.п. В противоположность этому для интерпретируемого кода в интерактивной среде интерактивная среда обрабатывает каждую переменную как строку. Поэтому интерактивные пользователи не могут определить тип для переменной.

Таким образом, существует необходимость в механизме для присвоения типов и других ограничений переменным в интерактивной среде.

Сущность изобретения

Предложенный механизм получает ограничения в интерактивной среде, ассоциирует эти ограничения с логическими структурами и затем применяет эти ограничения к логическим структурам при их появлении. Такие ограничения могут быть сохранены в метаданных, ассоциированных с соответствующей логической структурой. Ограничения могут определять тип данных для логической структуры, указатель предиката, указатель документирования, указатель синтаксического анализа, указатель генерирования данных, указатель подтверждения данных или указатель обработки и кодирования объекта. Эти ограничения являются расширяемыми для поддержки других указателей. Механизм позволяет интерактивным пользователям легко определять ограничения интерактивным способом.

Краткое описание чертежей

Фиг.1 - приведенное для примера вычислительное устройство, которое может использовать примерную среду административных инструментальных средств.

Фиг.2 - блок-схема, иллюстрирующая общее представление примерной базовой структуры административных инструментальных средств для предложенной среды административных инструментальных средств.

Фиг.3 - блок-схема, иллюстрирующая компоненты в составе специфических для ведущего узла (хоста) базовой структуры административных инструментальных средств, показанной на фиг.2.

Фиг.4 - блок-схема, иллюстрирующая компоненты в составе компонента механизма оболочки базовой структуры административных инструментальных средств, показанной на фиг.2.

Фиг.5 - пример структуры данных для определения команды “cmdlet”, подходящей для использования в базовой структуре административных инструментальных средств, показанной на фиг.2.

Фиг.6 - пример структуры данных для определения базового типа команды, из которого получена команда “cmdlet”, показанная на фиг.5.

Фиг.7 - другой пример структуры данных для определения базового типа команды, из которого получена команда “cmdlet”, показанная на фиг.5.

Фиг.8 - логическая блок-схема, иллюстрирующая пример процесса обработки в ведущем узле, которая выполняется в базовой структуре инструментальных средств, показанной на фиг.2.

Фиг.9 - логическая блок-схема, иллюстрирующая пример процесса обработки ввода, который выполняется в базовой структуре административных инструментальных средств, показанной на фиг.2.

Фиг.10 - логическая блок-схема, иллюстрирующая процесс обработки сценариев, подходящих для использования в процессе обработки вводов, показанном на фиг.9.

Фиг.11 - логическая блок-схема, иллюстрирующая процесс предварительной обработки, подходящий для использования в процессе обработки сценариев, показанном на фиг.10.

Фиг.12 - логическая блок-схема, иллюстрирующая процесс применения ограничений, подходящих для использования в процессе обработки сценариев, показанном на фиг.10.

Фиг.13 - функциональная блок-схема, иллюстрирующая процесс обработки командной строки в базовой структуре административных инструментальных средств, показанной на фиг.2.

Фиг.14 - логическая блок-схема, иллюстрирующая процесс обработки командных строк, подходящих для использования в процессе обработки вводов, показанном на фиг.9.

Фиг.15 - логическая блок-схема, иллюстрирующая пример процесса создания экземпляра команды “cmdlet”, подходящей для использования в обработке командных строк, показанной на фиг.14.

Фиг.16 - логическая блок-схема, иллюстрирующая пример процесса для заполнения свойств команды “cmdlet”, подходящей для использования в обработке команд, показанной на фиг.14.

Фиг.17 - логическая блок-схема, иллюстрирующая пример процесса исполнения команды “cmdlet”, подходящей для использования в обработке команд, показанной на фиг.14.

Фиг.18 - функциональная блок-схема примера администратора расширенного типа для использования в составе базовой структуры административных инструментальных средств, показанной на фиг.2.

Фиг.19 - графическое представление примерных последовательностей для команд “cmdlet” обработки вывода в конвейере.

Фиг.20 - пример обработки, выполняемой одной из команд “cmdlet” обработки вывода, показанных на фиг.19.

Фиг.21 - графическое представление приемной структуры для отображения информации, доступной при обработке согласно фиг.20.

Фиг.22 - таблица, представляющая пример синтаксиса для примерных команд “cmdlet” обработки вывода.

Фиг.23 - результаты, визуализируемые командой “cmdlet” вывода/консоли с использованием различных последовательностей конвейерной обработки для команд “cmdlet” обработки вывода.

Детальное описание предпочтительного варианта осуществления

Предложенный механизм получает ограничения в интерактивной среде и затем применяет эти ограничения к логическим структурам, вводимым в интерактивной

среде. Такие ограничения могут быть сохранены в метаданных, ассоциированных с соответствующей логической структурой. Ограничения могут определять тип данных для логической структуры, действительный диапазон для логической структуры и т.п. Механизм позволяет интерактивным пользователям легко определять ограничения интерактивным способом.

Последующее описание представляет конкретный пример административной среды инструментальных средств, в которой работает механизм. Другие примеры сред могут включать в себя признаки данного конкретного варианта осуществления и/или другие признаки, которые направлены на облегчение обработки ограничений в интерактивной среде.

Последующее детальное описание разделено на несколько разделов. Первый раздел описывает иллюстративную вычислительную среду, в которой может действовать среда административных инструментальных средств. Второй раздел описывает примерную базовую структуру для среды административных инструментальных средств. Последующие разделы описывают индивидуальные компоненты примерной базовой структуры и функционирование этих компонентов. Например, раздел «Примерная обработка сценариев» со ссылками на фиг.12 описывает примерный механизм для получения и применения ограничений в интерактивной среде.

Примерная вычислительная среда

Фиг.1 иллюстрирует пример вычислительного устройства, которое может быть использовано в примерной среде административных инструментальных средств. В самой базовой конфигурации вычислительное устройство 100 в типовом случае включает в себя, по меньшей мере, один блок 102 обработки и системную память 104. В зависимости от точной конфигурации и типа вычислительного устройства системная память может быть энергозависимой (RAM, ОЗУ), энергонезависимой (ROM, ПЗУ; флэш-память и т.д.) или некоторой комбинацией обоих указанных типов памяти. Системная память 104 в типовом случае включает в себя операционную систему 105, один или более программных модулей 106 и может включать в себя программные данные 107. Операционная система 105 включает в себя основанное на компонентах базовое средство 120 разработки, которое поддерживает компоненты (включая свойства и события), объекты, наследование, полиморфизм, отражение и обеспечивает объектно-ориентированный, основанный на компонентах интерфейс программирования приложений (API), такой как NET™ Framework, выпускаемый компанией Microsoft Corporation (Редмонт, шт.Вашингтон). Операционная система 105 также включает в себя базовое средство 200 разработки административных инструментальных средств, которое взаимодействует с основанным на компонентах базовым средством 120 разработки для поддержки разработки административных инструментальных средств (не показаны). Эта базовая конфигурация показана на фиг.1 компонентами, представленными в пределах пунктирного контура 108.

Вычислительное устройство 100 может содержать дополнительные свойства или функции. Например, вычислительное устройство 100 может включать в себя дополнительные устройства хранения данных (съемные и/или несъемные), например магнитные диски, оптические диски или магнитные ленты. Такие дополнительные запоминающие устройства иллюстрируются на фиг.1 съемным запоминающим устройством 109 и несъемным запоминающим устройством 110. Компьютерные носители для хранения данных могут включать в себя энергозависимые и энергонезависимые, съемные и несъемные носители, реализованные любым методом или по любой технологии для хранения информации, такой как машиночитаемые

команды, структуры данных, программные модули и другие данные. Системная память 104, съемное запоминающее устройство 109 и несъемное запоминающее устройство 110 являются примерами компьютерных носителей для хранения данных. Компьютерные носители для хранения данных включают в себя, без ограничения указанным, RAM, ROM, EEPROM (электронно-стираемое программируемое ПЗУ), флэш-память или другую технологию памяти, ПЗУ на компакт-дисках (CD-ROM), цифровые многоцелевые диски (DVD) или другие оптические ЗУ, магнитные кассеты, магнитные ленты, ЗУ на магнитных дисках или другие магнитные ЗУ, а также любой другой носитель, который может быть использован для хранения полезной информации и к которому может обращаться вычислительное устройство 100. Любой такой носитель для хранения данных может представлять собой часть устройства 100. Вычислительное устройство 100 может также иметь устройства 112 ввода, такие как клавиатура, мышь, перо, устройство речевого ввода, устройство сенсорного ввода и т.д. Также могут быть включены устройства 114 вывода, такие как дисплей, громкоговорители, принтер и т.д. Эти устройства хорошо известны в технике и не требуют более детального описания.

Вычислительное устройство также может содержать соединения 116 связи, которые позволяют устройству осуществлять связь с другими вычислительными устройствами 118, например, через сеть. Соединения 116 связи представляют собой один пример среды передачи. Среда передачи (коммуникационная среда) в типовом случае может быть реализована машиночитаемыми командами, структурами данных, программными модулями или иными данными в модулированном сигнале данных, таком как несущее колебание или иной транспортный механизм, и включает в себя любую среду доставки информации. Термин «модулированный сигнал данных» означает сигнал, в котором одна или более характеристик установлены или изменяются таким образом, чтобы кодировать информацию в сигнале. Например, без ограничений указанным, среда передачи включает в себя проводную среду передачи, такую как проводная сеть или прямое проводное соединение, и беспроводную среду передачи, такую как акустическая, радиочастотная, инфракрасная или иная беспроводная среда передачи. Термин «машиночитаемый носитель», как он использован в настоящем описании, включает в себя как среду хранения, так и среду передачи.

Иллюстративное базовое средство разработки административных инструментальных средств

На фиг.2 показана блок-схема, иллюстрирующая в общем виде приведенное для примера базовое средство разработки административных инструментальных средств. Базовое средство 200 разработки административных инструментальных средств включает в себя один или более ведущих (хост-) компонентов 202, специфических для хоста компонентов 204, независимых от хоста компонентов 206 и компонентов 208 обработчика. Независимые от хоста компоненты 206 могут осуществлять информационный обмен с каждым из других компонентов (т.е. хост-компонентов 202, специфических для хоста компонентов 204 и компонентов 208 обработчика). Каждый из этих компонентов кратко описан ниже и описан более детально, по мере необходимости, в последующих разделах.

Хост-компоненты 202 включают в себя одну или более программ хоста (например, программы 210-214 хоста), которые предоставляют свойства автоматической обработки для связанного приложения пользователям или другим программам. Каждая программа 210-214 хоста может предоставлять такие свойства

автоматической обработки в своем собственном конкретном стиле, например, посредством командной строки, графического пользовательского интерфейса (GUI), интерфейса распознавания речи, интерфейса программирования приложения (API), языка сценария, web-сервиса и т.п. Однако каждая из программ 210-214 хоста предоставляет одно или более свойств автоматической обработки через механизм, обеспеченный базовым средством разработки административных инструментальных средств.

В данном примере этот механизм использует команды “cmdlet” для того, чтобы показать возможности административных инструментальных средств пользователю связанных программ 210-214 хоста. Кроме того, этот механизм использует набор интерфейсов, сделанных доступными хостом, для помещения среды инструментальных средств в приложение, связанное с соответствующей программой 210-214 хоста. В последующем описании термин “cmdlet” используется для ссылки на команды, которые используются в иллюстративной среде инструментальных средств, описанной со ссылками на фиг.2-23.

Команды “cmdlet” соответствуют командам в традиционных административных средах. Однако команды “cmdlet” полностью отличаются от этих традиционных команд. Например, команды “cmdlet” в типовом случае меньше по размеру, чем их команды-двойники, поскольку команды “cmdlet” могут использовать общие функции, обеспечиваемые базовым средством разработки административных инструментальных средств, такие как синтаксический анализ, подтверждение данных, сообщение об ошибках и т.п. Поскольку такие общие функции могут быть реализованы однократно и протестированы однократно, использование команд “cmdlet” в пределах базового средства разработки административных инструментальных средств позволяет удержать дополнительные расходы на разработку и тестирование, связанные со специфическими для приложения функциями, на довольно низком уровне по сравнению с традиционными средами.

Кроме того, по сравнению с традиционными средами не требуется, чтобы команды “cmdlet” были автономно исполняемыми программами. Напротив, команды “cmdlet” могут исполняться в том же процессе в рамках базового средства разработки административных инструментальных средств. Это позволяет командам “cmdlet” обмениваться друг с другом активными объектами. Эта возможность обмениваться активными объектами позволяет командам “cmdlet” непосредственно вызывать методы по этим объектам. Детали создания и использования команд “cmdlet” описаны ниже более детально.

Рассматривая в общем виде, каждая программа 210-214 хоста управляет взаимодействиями между пользователем и другими компонентами в рамках базового средства разработки административных инструментальных средств. Эти взаимодействия могут включать в себя подсказки для параметров, сообщения об ошибках и т.п. В типовом случае программа 210-213 хоста может обеспечить свой собственный набор специфических для хоста команд “cmdlet” (например, команды “cmdlet” 219 хоста). Например, если программа хоста является программой электронной почты, то программа хоста может обеспечить команды “cmdlet”, которые взаимодействуют с почтовыми ящиками и сообщениями. Хотя на фиг.2 иллюстрируются программы 210-214 хоста, специалисту в данной области техники должно быть понятно, что компоненты 202 хоста могут включать в себя другие программы хоста, связанные с существующими и вновь создаваемыми приложениями. Эти другие программы хоста также будут вводить функции, обеспечиваемые средой

административных инструментальных средств, в связанные с ними приложения. Обработка, обеспечиваемая программой хоста, описана более детально ниже со ссылками на фиг.8.

В примерах, иллюстрируемых на фиг.2, программа хоста может представлять собой консоль управления (т.е. программу 210 хоста), которая обеспечивает простой, непротиворечивый пользовательский интерфейс администрирования для пользователей, чтобы создавать, сохранять и открывать административные инструментальные средства, которые управляют аппаратными средствами, программным обеспечением, сетевыми компонентами вычислительного устройства. Для выполнения этих функций программа 210 хоста обеспечивает набор сервисов для формирования GUI управления поверх базового средства разработки административных инструментальных средств. GUI-взаимодействия могут, таким образом, представляться как просматриваемые пользователем сценарии, которые помогают обучать пользователей возможностям сценариев, обеспечиваемым средой административных инструментальных средств.

В другом примере программа хоста может представлять собой интерактивную оболочку командной строки (т.е. программа 212 хоста). Интерактивная оболочка командной строки может обеспечить возможность ввода метаданных 216 оболочки в командную строку для воздействия на обработку, предусматриваемую командной строкой.

Еще в одном примере программа хоста может представлять собой web-сервис (например, программа 214 хоста), который использует спецификации промышленных стандартов для распределенных вычислений и взаимодействия между платформами, языков программирования и приложений.

В дополнение к этим примерам третьи стороны могут добавить свои собственные компоненты хостов путем создания интерфейсов «третьей стороны» или «провайдера» и команд “cmdlet” провайдера, которые используются с их программой хоста или с другими программами хоста. Интерфейс провайдера показывает приложение или инфраструктуру, так что этим приложением или инфраструктурой можно манипулировать с использованием базового средства разработки административных инструментальных средств. Команды “cmdlet” провайдера обеспечивают средства автоматической обработки для навигации, диагностики, конфигурирования, установления рабочего цикла, операций и т.п. Комнды “cmdlet” провайдера проявляют полиморфное поведение команды “cmdlet” на полностью гетерогенном наборе хранилищ данных. Среда административных инструментальных средств оперирует над командами “cmdlet” провайдера с тем же приоритетом, что и с другими классами команд “cmdlet”. Команда “cmdlet” провайдера создается с использованием тех же механизмов, что и в случае других команд “cmdlet”. Команды “cmdlet” провайдера проявляют конкретную функциональность приложения или инфраструктуру по отношению к базовому средству разработки административных инструментальных средств. Таким образом, путем использования команд “cmdlet” разработчикам продуктов необходимо только создавать один компонент хоста, который позволит затем их продукту работать с множеством административных инструментальных средств. Например, в приведенной для примера среде административных инструментальных средств меню помощи графического пользовательского интерфейса системного уровня могут быть интегрированы и перенесены в существующие приложения.

Специфические для хоста компоненты

Специфические для хоста компоненты 204 включают в себя набор сервисов, которые вычислительные системы (например, вычислительное устройство 100 на фиг.1) используют для изоляции базового средства разработки административных инструментальных средств от специфики платформы, на которой работает указанное базовое средство разработки. Таким образом, имеется набор специфических для хоста компонентов для каждого типа платформы. Специфические для хоста компоненты позволяют пользователям использовать одни и те же административные инструментальные средства на различных операционных системах.

В соответствии с фиг.2 специфические для хоста компоненты 204 могут включать в себя компонент 302 интеллектуального восприятия/доступа к метаданным, компонент 304 “cmdlet” помощи, компонент 306 конфигурирования/регистрации, компонент 308 установки “cmdlet” и компонент 309 интерфейсов вывода. Компоненты 302-308 осуществляют информационный обмен с администратором 312 хранилища баз данных, связанным с хранилищем 314 баз данных. Синтаксический анализатор 220 и механизм 222 сценариев осуществляют информационный обмен с компонентом 302 интеллектуального восприятия/доступа к метаданным. Механизм 224 оболочки осуществляет информационный обмен с компонентом 304 “cmdlet” помощи, компонентом 306 конфигурирования/регистрации, компонентом 308 установки “cmdlet” и компонентом 309 интерфейсов вывода. Компонент 309 интерфейсов вывода включает в себя интерфейсы, обеспечиваемые хостом для “cmdlets” вывода. Эти “cmdlets” вывода могут затем вызвать объект вывода хоста для выполнения визуализации. Специфические для хоста компоненты 204 могут также включать в себя компонент 310 регистрации/проверки, используемый механизмом 224 оболочки для обеспечения информационного обмена со специфическими для хоста (то есть специфическими для платформы) сервисами, которые обеспечивают возможности регистрации и проверки.

В одном приведенном для примера базовом средстве разработки административных инструментальных средств компонент 302 интеллектуального восприятия/доступа к метаданным обеспечивает автоматическое завершение команд, параметров и значений параметров. Компонент 304 “cmdlet” помощи обеспечивает настроенную систему помощи на основе пользовательского интерфейса хоста.

Компоненты обработчика

Согласно фиг.2 компоненты 208 обработчика включают в себя унаследованные утилиты 230, управляющие “cmdlets” 232, неуправляющие “cmdlets” 234, “cmdlets” 236 удаленного действия и интерфейс 238 web-сервисов. Управляющие “cmdlets” 232 (также упоминаемые как “cmdlets” платформы) включают в себя “cmdlets”, которые запрашивают или манипулируют информацией конфигурирования, связанной с вычислительным устройством. Поскольку управляющие “cmdlets” 232 манипулируют информацией системного типа, они являются зависимыми от конкретной платформы. Однако каждая платформа в типовом случае имеет управляющие “cmdlets” 232, которые обеспечивают действия, сходные с соответствующими действиями управляющих “cmdlets” 232 на других платформах. Например, каждая платформа поддерживает управляющие “cmdlets” 232, которые получают и устанавливают системные атрибуты администрирования (например, получить/процесс, установить/IP-адрес). Не зависимые от платформы компоненты 206 осуществляют связь с управляющими “cmdlets” через объекты “cmdlets”, генерируемые в не зависимых от платформы компонентах 206. Примерные структуры данных для объектов “cmdlets” описаны более детально ниже со ссылками на фиг.5-7.

Неуправляющие “cmdlets” 234 (иногда упоминаемые как базовые “cmdlets”) включают в себя “cmdlets”, которые группируют, сортируют, фильтруют и выполняют другую обработку на объектах, обеспечиваемых управляющими “cmdlets” 232. Неуправляющие “cmdlets” 234 могут также включать в себя “cmdlets” для форматирования и вывода данных, связанных с объектами конвейерной обработки. Примерный механизм для обеспечения вывода управляемой данными командной строки описан ниже со ссылкой на фиг.19-23. Неуправляющие “cmdlets” 234 могут быть одинаковыми на каждой платформе и обеспечивать набор утилит, которые взаимодействуют с не зависимыми от хоста компонентами 206 через объекты “cmdlets”. Взаимодействия между неуправляющими “cmdlets” 234 и не зависимыми от хоста компонентами 206 обеспечивают возможность отражения на объекты и возможность обработки на отраженных объектах независимо от их (объектов) типа. Таким образом, эти утилиты позволяют разработчикам писать неуправляющие “cmdlets” однократно и затем применять эти неуправляющие “cmdlets” по всем классам объектов, поддерживаемых на вычислительной системе. В прошлом разработчики должны были сначала осмыслить формат данных, которые должны были обрабатываться, и затем писать приложение для обработки только этих данных. Следовательно, традиционные приложения могли только обрабатывать данные очень ограниченного объема. Пример механизма для обработки объектов независимо от их типа объекта описан ниже со ссылкой на фиг.18.

Унаследованные утилиты 230 включают в себя существующие выполняемые программы, такие как win32, которые исполняются под cmd.exe. Каждая унаследованная утилита 230 осуществляет связь с базовым средством разработки административных инструментальных средств (то есть stdin и stdout), которые являются типом объекта в базовой структуре объекта. Поскольку унаследованные утилиты 230 используют текстовые потоки, то основанных на отражении операций, обеспечиваемых базовым средством разработки административных инструментальных средств, не имеется. Унаследованные утилиты 230 исполняются в процессе, отличающемся от базового средства разработки административных инструментальных средств. Хотя не показано, другие “cmdlets” также могут действовать вне процесса.

“cmdlets” 236 удаленного действия и интерфейс 238 web-сервисов обеспечивают механизмы удаленного действия для доступа к интерактивным программируемым средам административных инструментальных средств на других вычислительных устройствах через среду передачи, такую как Интернет или интранет (например, Интернет/интранет 240, показанные на фиг.2). В иллюстративном базовом средстве разработки административных инструментальных средств механизмы удаленного действия поддерживают объединенные сервисы, которые зависят от инфраструктуры, которая охватывает множество независимых областей управления. Механизм удаленного действия позволяет исполнять сценарии на одном или множестве удаленных вычислительных устройств. Сценарии могут исполняться на одной или множестве удаленных систем. Результаты сценариев могут обрабатываться, когда каждый индивидуальный сценарий завершается, или результаты могут агрегироваться и обрабатываться совместно после того, как будут завершены все сценарии на различных вычислительных устройствах.

Например, web-сервис 214, показанный как один из компонентов 202 хоста, может являться удаленным агентом. Удаленный агент обрабатывает представление удаленных запросов команд к синтаксическому анализатору и базовому средству

разработки административных инструментальных средств на целевой системе. Удаленные “cmdlets” служат в качестве удаленного клиента для обеспечения доступа к удаленному агенту. Удаленный агент и удаленные “cmdlets” осуществляют связь посредством анализируемого потока. Этот анализируемый поток может быть защищен на протокольном уровне, или дополнительные “cmdlets” могут быть использованы для шифрования и затем дешифрования анализируемого потока.

Не зависимые от хоста компоненты

Не зависимые от хоста компоненты 206 включают в себя синтаксический анализатор 220, механизм 222 сценариев и механизм 224 оболочки. Не зависимые от хоста компоненты 206 обеспечивают механизмы и сервисы для группирования множества “cmdlets”, координирования действия “cmdlets” и координирования взаимодействия других ресурсов, сессий и заданий с “cmdlets”.

Примерный синтаксический анализатор

Синтаксический анализатор 220 обеспечивает механизмы для приема входных запросов от различных программ хоста и отображения входных запросов на однородные объекты “cmdlet”, которые используются во всем базовом средстве разработки административных инструментальных средств, например в механизме 224 оболочки. Кроме того, синтаксический анализатор 220 может выполнять обработку данных на основе принятого ввода. Пример способа выполнения обработки данных на основе ввода описан ниже со ссылкой на фиг.12. Синтаксический анализатор 220 базового средства разработки административных инструментальных средств обеспечивает возможность простого представления различных языков или синтаксиса пользователям для одних и тех же возможностей. Например, ввиду того что синтаксический анализатор 220 выполняет интерпретацию входных запросов, изменение в коде в синтаксическом анализаторе 220, которое затрагивает ожидаемый синтаксис ввода, по существу будет влиять на каждого пользователя базового средства разработки административных инструментальных средств. Поэтому системные администраторы могут обеспечить различные синтаксические анализаторы на разных вычислительных устройствах, которые поддерживают различный синтаксис. Однако каждый пользователь, работающий с тем же самым синтаксическим анализатором, будет воспринимать согласованный синтаксис для каждой команды “cmdlet”. В противоположность этому, в традиционных средах каждая команда реализовывала свой собственный синтаксис. Таким образом, в условиях тысяч команд, когда каждая среда поддерживает множество различных синтаксисов, обычно многие из них бли не согласованными друг с другом.

Примерный механизм сценариев

Механизм 222 сценариев обеспечивает методы и сервисы для связывания множества “cmdlets” вместе с использованием сценария. Сценарий является объединением командных строк, которые совместно используют состояние сессии по строгим правилам наследования. Множество командных строк в сценарии могут исполняться синхронно или асинхронно на основе синтаксиса, обеспечиваемого во входном запросе. Механизм 222 сценария имеет возможность обрабатывать структуры управления, такие как циклы и условные предложения, и обрабатывать переменные в сценарии. Механизм сценария также управляет состоянием сессии и обеспечивает “cmdlets”-доступ к данным сессии на основе стратегии (не показано).

Примерный механизм оболочки

Механизм 224 оболочки обеспечивает обработку “cmdlets”, идентифицированных синтаксическим анализатором 220. На фиг.4 показан примерный механизм 224

оболочки в составе базового средства 200 разработки административных инструментальных средств. Примерный механизм 224 оболочки содержит конвейерный процессор 402, загрузчик 404, процессор 406 метаданных, обработчик 408 ошибок и событий, администратор 410 сессии и администратор 412 расширенного типа.

Примерный процессор метаданных

Процессор 406 метаданных конфигурирован для доступа и сохранения метаданных в ЗУ метаданных, таком как хранилище 314 базы данных, показанное на фиг.3. Метаданные могут поставляться посредством командной строки, внутри определения класса “cmdlet” и т.п. Различные компоненты в составе базового средства 200 разработки административных инструментальных средств могут запросить метаданные при выполнении их обработки. Например, синтаксический анализатор 202 может запросить метаданные для подтверждения параметров, поданных в командной строке.

Процессор 408 ошибок и событий обеспечивает объект ошибки для сохранения информации о каждом появлении ошибки в процессе обработки командной строки. Дополнительная информация о конкретном процессоре ошибок и событий, особенно подходящем для настоящего базового средства 200 разработки административных инструментальных средств, содержится в патентной заявке США №10/413054, патент США №20040205415 на «Систему и способ для сохранения информации об ошибках в среде командных строк», того же заявителя, что и в настоящем изобретении, которая включена в настоящее описание посредством ссылки.

Примерный администратор сессии

Администратор 410 сессии поставляет информацию о сессии и состоянии другим компонентам в базовом средстве 200 разработки административных инструментальных седств. К информации состояния, которой управляет администратор сессии, может обеспечиваться доступ для “cmdlet”, хоста или механизма оболочки через интерфейсы программирования. Эти интерфейсы программирования обеспечивают возможность создания, модифицирования и удаления информации состояния.

Примерный конвейерный процессор и загрузчик

Загрузчик 404 конфигурируется для загрузки каждой команды “cmdlet” в память для исполнения “cmdlet” конвейерным процессором 402. Конвейерный процессор 402 включает в себя “cmdlet”-процессор 420 и “cmdlet”-администратор 422. “cmdlet”-процессор 420 диспетчеризует отдельные “cmdlets”. Если “cmdlet” требует исполнения на удаленной машине или на множестве удаленных машин, то “cmdlet”-процессор 420 координирует исполнение с “cmdlet” 236 удаленной обработки, показанным на фиг.2. “cmdlet”-администратор 422 управляет исполнением совокупностей “cmdlets”. “cmdlet”-администратор 422, “cmdlet”-процессор 420 и механизм 222 сценария (фиг.2) осуществляют связь друг с другом, чтобы выполнять обработку ввода, полученного от программы 210-214 хоста. Информационный обмен может быть рекурсивным по своему характеру. Например, если программа хоста обеспечивает сценарий, то сценарий может вызвать “cmdlet”-администратор 422 для исполнения “cmdlet”, что само может представлять сценарий. Сценарий может тогда исполняться механизмом 222 сценария. Иллюстративная процедура обработки для механизма оболочки описана в деталях в связи с фиг.14.

Примерный администратор расширенного типа

Как упомянуто выше, базовое средство 200 разработки административных

инструментальных средств обеспечивает набор утилит, что обеспечивает отражение объектов и дает возможность обработки на отраженных объектах независимо от их (объекта) типа. Базовое средство 200 разработки административных инструментальных средств взаимодействует с базовой структурой компонента на вычислительной системе (базовая структура 120 компонента на фиг.1) для выполнения этого отражения. Специалисту в данной области техники должно быть понятно, что отражение обеспечивает возможность запроса объекта и получения типа для объекта и затем отражения на различные объекты и свойства, связанные с данным типом объекта, чтобы получить другие объекты и/или желательное значение.

Даже хотя отражение обеспечивает базовое средство 200 разработки административных инструментальных средств значительным объемом информации по объектам, изобретатели исходили из того, что отражение фокусируется на типе объекта. Например, при отражении таблицы данных базы данных возвращаемая информация заключается в том, что база данных имеет два свойства: сойство столбца и свойство строки. Эти два свойства не обеспечивают достаточно информации относительно «объектов» в таблице данных. Аналогичные проблемы возникают, когда используется отражение на расширяемый язык разметки (XML) и другие объекты.

Поэтому изобретатели сосредоточились на администраторе 412 расширенного типа, который фокусируется на использовании типа. Для этого администратора расширенного типа тип объекта не принципиален. Вместо этого администратор расширенного типа концентрируется на том, может ли объект использоваться для получения требуемой информации. Для вышеприведенного примера с таблицей данных изобретатели исходили из того, что информация о том, что таблица данных имеет свойство столбцов и свойство строк, не представляет особого интереса, а интерес представляет информация, содержащаяся в одном столбце. Фокусируясь на использовании, можно ассоциировать каждую строку с «объектом» и ассоциировать каждый столбец со «свойством» этого «объекта». Таким образом, администратор 412 расширенного типа обеспечивает механизм для создания «объектов» из любого типа точно синтаксически анализируемого ввода. При этом администратор 412 расширенного типа поставляет средства отражения, обеспечиваемые базовой структурой 120 на основе компонента, и распространяет «отражение» на любой тип точно синтаксически анализируемого ввода.

В общих чертах администратор расширенного типа конфигурируется для доступа к точно синтаксически анализируемому вводу (не показано) и для коррелирования точно синтаксически анализируемого ввода с запрошенным типом данных. Администратор 412 расширенного типа выдает затем запрошенную информацию запрашивающему компоненту, такому как конвейерный процессор 402 или синтаксический анализатор 220. В последующем описании точно синтаксически анализируемый ввод определяется как ввод, в котором можно различить свойства и значения. Некоторые точно синтаксически анализируемые вводы включают в себя вводы, соответствующие Windows Management Instrumentation (WMI), ActiveX Data Object (ADO), eXtensible Markup Language (XML), и объектный ввод, такой как объекты .NET. Другие точно синтаксически анализируемые вводы могут включать в себя форматы данных третьей стороны.

На фиг.18 показана функциональная блок-схема иллюстративного администратора расширенного типа для использования в базовом средстве разработки административных инструментальных средств. Для целей объяснения функции

(обозначенные цифрой «3» в кружке), обеспечиваемые администратором расширенного типа, контрастируют с функциями, обеспечиваемыми традиционной сильно связанной системой (обозначены цифрой «1» в ружке), и с функциями, обеспечиваемыми системой отражения (обозначены цифрой «2» в кружке). В традиционной сильно связанной системе вызывающий оператор 1802 в приложении непосредственно обращается к информации (например, свойствам P1 и P2, методам M1 и M2) в объекте A. Как упомянуто выше, вызывающий оператор 1802 должен априорно знать свойства (например, свойства P1 и P2) и методы (например, методы M1 и M2), обеспечиваемые объектом A во время компилирования. В системе отражения универсальный код 1820 (не зависящий от какого-либо типа данных) запрашивает систему 1808, которая выполняет отражение 1810 на запрашиваемом объекте и возвращает информацию (например, свойства P1 и P2, методы M1 и M2) об объекте (например, объекте A) в универсальный код. Хотя для объекта A не показано, возвращаемая информация может включать в себя дополнительную информацию, такую как данные поставщика, файл, дату и т.д. Таким образом, посредством отражения универсальный код 1820 получает, по меньшей мере, ту же информацию, что и обеспечиваемую сильно связанной системой. Система отражения также позволяет вызывающему оператору 1802 запрашивать систему и получать дополнительную информацию без априорного знания о параметрах.

Как в сильно связанных системах, так и в системах отражения новые типы данных не могут быть просто включены в операционную среду. Например, в сильно связанной системе, после того как операционная среда доставлена, эта операционная среда не может внедрять новые типы данных, поскольку ее нужно было бы перестроить для обеспечения их поддержки. Аналогичным образом, в системе отражения метаданные для каждого класса объекта являются фиксированными. Таким образом, внедрение типов данных обычно не делается.

Однако с использованием представленного администратора расширенного типа новые типы данных могут быть внедрены в операционную систему. В случае администратора 1822 расширенного типа универсальный код 1820 может отражаться на запрашиваемый объект, чтобы получить расширенные типы данных (например, объект A'), обеспечиваемые различными внешними источниками, такими как объекты третьей стороны (например, объект A' и B), семантический web-сервис 1832, онтологический сервис 1834 и т.п. Как показано, объект третьей стороны может расширять существующий объект (например, объект A') или может создавать полностью новый объект (например, объект B).

Каждый из этих внешних источников может регистрировать свою уникальную структуру в метаданных 1840 типа и может обеспечивать код 1842. Когда объект запрашивается, администратор 1840 расширенного типа просматривает метаданные 1840 типа для определения того, зарегистрирован ли данный объект. Если объект не зарегистрирован в метаданных 1840 типа, то выполняется отражение. В ином случае выполняется расширенное отражение. Код 1842 возвращает дополнительные свойства и методы, связанные с типом, для которого выполнено отражение. Например, если входным типом является XML, то код 1842 может включать файл описания, который описывает способ, которым используется XML для создания объектов из XML-документа. Таким образом, метаданные 1840 типа описывают, каким образом администратор 412 расширенного типа должен запрашивать различные типы точно синтаксически анализируемого ввода (например, объекты A' и B третьей стороны, семантический web-сервис 1832) для получения

желательных свойств для создания объекта для данного конкретного типа ввода, и код 1842 обеспечивает инструкции для получения этих желательных свойств. В результате администратор 412 расширенного типа обеспечивает уровень косвенности, который позволяет «отражать» все типы объектов.

В дополнение к обеспечению расширенных типов администратор 412 расширенного типа обеспечивает дополнительные механизмы запроса, такие как механизм пути свойства, механизм ключа, механизм сравнения, механизм преобразования, механизм подстановки, механизм установки свойств, механизм отношений и т.п. Каждый из этих механизмов запроса, описанный ниже в разделе «Пример обработки администратора расширенного типа», обеспечивает гибкость для системных администраторов при вводе командных строк. Для реализации семантики для администратора расширенного типа могут использоваться различные методы. Ниже описано три метода. Однако специалистам в данной области техники должно быть понятно, что могут использоваться варианты этих методов без отклонения от объема заявленного изобретения.

В одном методе может быть обеспечен ряд классов, имеющих статические методы (например, getproperty() («получить свойство»)). Объект вводится в статический метод (например, getproperty(object) («получить свойство (объекта)»), и статический метод возвращает набор результатов. В другом методе операционная среда инкапсулирует объект адаптером. Таким образом, никакой ввод не подается. Каждый экземпляр адаптера имеет метод getproperty, который воздействует на инкапсулированный объект и возвращает свойства для инкапсулированного объекта. Данный метод иллюстрируется следующим псевдокодом:

```
Class Adaptor
{
        Object X;
        getProperties();
}.
```

Еще в одном методе класс адаптера осуществляет субклассификацию объекта. Традиционно, субкассификация происходит перед компилированием. Однако в некоторых операционных средах субклассификация может производиться динамически. Для этих типов сред следующий псевдокод иллюстрирует указанный метод:

```
Class Adaptor : A
{
        getProperties()
        {
                return data;
        }
 }.
```

Таким образом, как показано на фиг.18, администратор расширенного типа позволяет разработчикам создавать новый тип данных, регистрировать тип данных и позволяет другим приложениям и “cmdlets” использовать новый тип данных. В

противоположность этому в известных средах администрирования каждый тип данных должен был быть известным в момент компилирования, чтобы можно было непосредственно обращаться к свойству или методу, связанным с объектом, реализованным из этого типа данных. Поэтому добавление новых типов данных, которые поддерживались бы средой администрирования, в прошлом осуществлялось редко.

Согласно фиг.2 в общем представлении базовое средство 200 разработки административных инструментальных средств не основывается на оболочке для координирования исполнения команд, вводимых пользователем, а вместо этого расщепляет функции на часть обработки (например, не зависимые от хоста компоненты 206) и части пользовательского взаимодействия (например, “cmdlets” хоста). Кроме того, представленная среда административных инструментальных средств в значительной степени упрощает программирование административных инструментальных средств, поскольку код, требуемый для синтаксического анализа и подтверждения данных, больше не включается в каждую команду, а вместо этого обеспечивается компонентами (например, синтаксическим анализатором 220) в базовом средстве разработки административных инструментальных средств. Пример обработки, выполняемой в базовом средстве разработки административных инструментальных средств, описан ниже.

Пример функционирования

Фиг.5-7 графически иллюстрируют примеры структур данных, используемых в среде административных инструментальных средств. Специалисту в данной области техники должно быть понятно, что определенная обработка может быть выполнена компонентом, отличающимся от описанных ниже, без отклонения от объема настоящего изобретения. Перед описанием обработки, выполняемой в компонентах среды административных инструментальных средств, ниже будут представлены примерные структуры данных, используемые в базовом средстве разработки административных инструментальных средств.

Примеры структур данных для объектов “cmdlet”

На фиг.5 представлена структура данных для определения объекта “cmdlet”, подходящего для использования в базовом средстве разработки административных инструментальных средств, показанном на фиг.2. При выполнении “cmdlet” может быть управляющим “cmdlet”, неуправляющим “cmdlet”, “cmdlet” хоста, “cmdlet” поставщика и т.п. Последующее описание представляет создание “cmdlet” с точки зрения системного администратора (т.е. “cmdlet” поставщика). Однако каждый тип “cmdlet” создается одинаковым путем и работает сходным образом. “cmdlet” может быть написан на любом языке, таком как C#. Кроме того, “cmdlet” может быть написан с использованием языка сценариев и т.п. Если среда административных инструментальных средств действует с .NET.Framework, то “cmdlet” будет объектом .NET.

“cmdlet” 500 поставщика (далее упоминаемый как “cmdlet” 500) является общедоступным классом, имеющим имя “cmdlet” класса (например, StopProcess 500). “cmdlet” 500 выводится из “cmdlet” класса 506. Пример структуры данных для “cmdlet” класса 506 описан ниже со ссылкой на фиг.6. Каждый “cmdlet” 500 связан с командным атрибутом 502, который ассоциирует имя (например, Stop/Process) с “cmdlet” 500. Имя зарегистрировано в среде административных инструментальных средств. Как описано ниже, синтаксический анализатор осуществляет поиск в регистре “cmdlet” для идентификации “cmdlet” 500, когда командная строка, имеющая имя

(например, Stop/Process), подается в качестве ввода в командную строку или в сценарий.

“cmdlet” 500 ассоциирован с механизмом грамматики, который определяет грамматику для ожидаемых параметров ввода для данного “cmdlet”. Механизм грамматики может быть прямо или косвенно связан с “cmdlet”. Например, “cmdlet” 500 иллюстрирует прямую грамматическую связь. В этом “cmdlet” 500 декларированы один или более общедоступных параметров (например, ProcessName («имя процесса») 510 и PID («идентификатор процесса») 512). Декларация общедоступных параметров вызывает синтаксический анализ объектов ввода для “cmdlet” 500. Альтернативно, описание параметров может появиться во внешнем источнике, таком как XML-документ. Описание параметров в этом внешнем источнике будет затем управлять синтаксическим анализом объектов ввода для “cmdlet”.

Каждый общедоступный параметр 510, 512 может иметь один или более атрибутов (например, указателей (директив)), связанных с ним. Указатели могут быть из любой из следующих категорий: указатель 521 синтаксического анализа, указатель 522 подтверждения данных, указатель 523 генерации данных, указатель 524 обработки, указатель 525 кодирования и указатель 526 документирования. Указатели могут быть взяты в квадратные скобки. Каждый указатель описывает операцию, подлежащую выполнению над параметром следующего ожидаемого ввода. Некоторые из диектив могут также применяться на уровне класса, такие как указатели типа пользовательского взаимодействия. Указатели сохраняются в метаданных, связанных с “cmdlet”. Применение этих атрибутов описано ниже со ссылками на фиг.12.

Эти атрибуты также могут влиять на заполнение параметров, декларированных в “cmdlet”. Иллюстративный процесс заполнения этих параметров описан ниже со ссылкой на фиг.16. Механизм оболочки может применить эти указатели для обеспечения совместимости. “cmdlet” 500 включает в себя первый метод 530 (далее упоминаемый также как метод 530 StartProcessing («начало обработки»)) и второй метод 540 (далее упоминаемый также как метод 540 processRecord («запись процесса»)). Механизм оболочки использует первый и второй методы 530, 540 для управления обработкой “cmdlet” 500. Например, первый метод 530 выполняется однократно и выполняет функции установки. Код 542 во втором методе 540 выполняется для каждого объекта (например, запись), который требуется для обработки посредством “cmdlet” 500. “cmdlet” 500 также может включать в себя третий метод (не показан), который заканчивает обработку после “cmdlet” 500.

Таким образом, как показано на фиг.5, код 542 во втором методе 540 в типовом случае довольно краток и не содержит функций, требуемых в традиционной среде административных инструментальных средств, таких как код синтаксического анализа, код подтверждения данных и т.п. Таким образом, системные администраторы могут разрабатывать сложные административные задачи без изучения сложного языка программирования.

На фиг.6 представлен пример структуры 600 данных для определения базового класса 602 “cmdlet”, из которого выводится “cmdlet”, показанный на фиг.5. Базовый класс 602 “cmdlet” включает в себя инструкции, которые обеспечивают дополнительные функции, всякий раз, когда “cmdlet” включает в себя оператор подключения программы и соответствующий переключатель вводится в командную строку или в сценарий (совместно упоминаемые как командный ввод).

Приведенный пример структуры 600 данных включает в себя параметры, такие как словесный Булев параметр 610, параметр «whatif» («что если») 620 и

подтверждение 630. Как описано ниже, эти параметры соответствуют строкам, которые могут быть введены в командном вводе. Пример структуры 600 данных может также включать в себя метод 640 защиты, который определяет, разрешено ли исполнение запрошенной задачи.

На фиг.7 представлен другой пример структуры 700 данных для определения “cmdlet”. В общем представлении структура 700 данных обеспечивает средство для выражения в явном виде договора между базовым средством разработки административных инструментальных средств и “cmdlet”. Подобно структуре 500 данных, структура 700 данных является общедоступным классом, который выводится из класса 704 “cmdlet”. Разработчик программного обеспечения определяет декларацию 702 “cmdlet”, которая связывает пару объект/действие, такую как «получить/процесс» и «форматировать/таблица», с “cmdlet” 700. Пара объект/действие регистрируется в среде административных инструментальных средств. Объект или действие могут быть неявно выражены в имени “cmdlet”. Также, подобно структуре 500 данных, структура 700 данных может включать в себя один или более общедоступных членов (например, имя 730, Ресурс 732), которые могут быть связаны с одним или более указателей 520-526, описанных в связи со структурой 500 данных.

Однако в этом примере 700 структуры данных каждый из ожидаемых параметров ввода 730, 732 связан с атрибутом 731 и 733 ввода соответственно. Атрибуты 731 и 733 ввода определяют, что данные для их соответствующих параметров 730, 732 должны быть определены из командной строки. Таким образом, в этом примере структуры 700 данных отсутствуют ожидаемые параметры ввода, которые пополняются из конвейерного объекта, который был генерирован другим “cmdlet”. Таким образом, структура 700 данных не переопределяет первый метод (например, Начало Обработки) или второй метод (например, Запись Обработки), которые обеспечены базовым классом “cmdlet”.

Структура 700 данных также может включать скрытый член 740, который не распознается как параметр ввода. Скрытый член 740 может использоваться для хранения данных, которые генерируются на основе одного из указателей.

Таким образом, как показано в структуре 700 данных, посредством использования декларирования общедоступных свойств и указателей в конкретном классе “cmdlet” разработчики “cmdlet” могут просто определять грамматику для ожидаемых параметров ввода для разрабатываемых “cmdlet” и определять обработку, которая должна выполняться над ожидаемыми параметрами ввода, без необходимости для разработчиков “cmdlet” генерировать какую-либо базовую логику. Структура 700 данных иллюстрирует прямую связь между “cmdlet” и механизмом грамматики. Как упомянуто выше, эта связь может также быть косвенной, такой как через задание определений ожидаемых параметров во внешнем источнике, таком как XML-документ.

Ниже описаны примеры последовательности обработки в среде административных инструментальных средств.

Пример последовательности обработки хоста

На фиг.8 показана блок-схема логической последовательности, иллюстрирующая примерный процесс обработки хоста, которая выполняется в базовом средстве разработки административных инструментальных средств, показанном на фиг.2. Процесс 800 начинается в блоке 801, где принимается запрос на инициирование среды административных инструментальных средств для конкретного приложения. Запрос может посылаться локально посредством клавишного ввода, например выбором пиктограммы приложения, или дистанционно посредством интерфейса web-сервиса

или другого вычислительного устройства. Для любого сценария обработка продолжается в блоке 802.

В блоке 8023 конкретное приложение (например, программа хоста) на «целевом» вычислительном устройстве устанавливает свою среду. Это включает в себя определение того, какие поднаборы “cmdlets” (например, управляющие “cmdlets” 232, неуправляющие “cmdlets” 234, “cmdlets” 218 хоста) делаются доступными пользователю. В типовом случае программа хоста будет делать доступными все неуправляющие “cmdlets” 234 и все свои собственные “cmdlets” 218 хоста. Кроме того, программа хоста будет делать доступными поднабор управляющих “cmdlets” 234, таких как “cmdlets”, имеющих отношение к процессам, диску и т.п. Таким образом, как только программа хоста сделала доступным поднабор “cmdlets”, базовое средство разработки административных инструментальных средств эффективно вводится в соответствующее приложение. Обработка продолжается в блоке 804.

В блоке 804 посредством конкретного приложения осуществляется ввод. Как упомянуто выше, ввод может иметь различные формы, включая командные строки, сценарии, речевой ввод, GUI и т.п. Например, когда ввод получают посредством командной строки, такой ввод извлекается из нажатий клавиш на клавиатуре. Для хоста GUI строка составляется на основе GUI. Обработка продолжается в блоке 806.

В блоке 806 ввод выдается в другие компоненты в составе базового средства разработки административных инструментальных средств для обработки. Программа хоста может направлять ввод непосредственно к другим компонентам, таким как синтаксический анализатор. Альтернативно, программа может направлять ввод через один из ее “cmdlets” хоста. “cmdlet” хоста может преобразовывать свой конкретный тип ввода (например, речевого) в тип ввода (например, текстовая строка, сценарий), который распознается базовым средством разработки административных инструментальных средств). Например, речевой ввод может быть преобразован в сценарий или в командную строку в зависимости от содержания речевого ввода. Поскольку каждая программа хоста несет ответственность за преобразование своего типа ввода во ввод, распознаваемый базовым средством разработки административных инструментальных средств, это базовое средство разработки административных инструментальных средств может принимать ввод от любого числа различных компонентов хоста. Кроме того, базовое средство разработки административных инструментальных средств обеспечивает обогащенный набор утилит, который выполняет преобразование между типами данных, когда ввод направляется через один из его “cmdlets”. Обработка, выполняемая над вводом другими компонентами, описана ниже со ссылками на другие чертежи. Обработка хоста продолжается в решающем блоке 808.

В решающем блоке 808 принимается решение, был ли принят запрос на дополнительный ввод. Это может произойти, если одному из других компонентов, несущих ответственность за обработку ввода, требуется дополнительная информация от пользователя, чтобы завершить свою обработку. Например, может потребоваться пароль для доступа к некоторым данным, может потребоваться подтверждение конкретных действий и т.п. Для некоторых типов программ хоста (например, речевой почты) запрос такой, как этот, может быть несоответствующим. Таким образом, вместо запроса у пользователя дополнительной информации программа хоста может упорядочить (преобразовать в последовательную форму) состояние, приостановить состояние и послать уведомление, чтобы в более позднее время состояние можно было возобновить и продолжить исполнение ввода. В другом варианте программа хоста

может обеспечить значение по умолчанию по истечении предварительно определенного интервала времени. Если запрос на дополнительный ввод принят, то цикл обработки возвращается назад к блоку 804, где получается дополнительный ввод. Затем обработка продолжается в блоках 806 и 808, как описано выше. Если не принимается запрос на дополнительный ввод и ввод обработан, то обработка продолжается в блоке 810.

В блоке 810 принимаются результаты от других компонентов в базовом средстве разработки административных инструментальных средств. Результаты могут включать сообщения об ошибках, статус и т.п. Результаты представляются в форме объекта, который распознается и обрабатывается “cmdlet” хоста в базовом средстве разработки административных инструментальных средств. Как описано ниже, код, написанный для каждого “cmdlet” хоста, является минимальным. Таким образом, обогащенный набор вывода может быть отображен без необходимости больших капиталовложений в затраты на разработку. Обработка продолжается в блоке 812.

В блоке 812 результаты могут быть просмотрены. “cmdlet” хоста преобразует результаты в стиль отображения, поддерживаемый программой хоста. Например, возвращенный объект может быь отображен посредством программы хоста графического пользовательского интерфейса (GUI) с использованием графического представления, такого как пиктограмма, лающая собака и т.п. Формат и вывод по умолчанию могут использовать примерные “cmdlet” обработки вывода, описанные ниже со ссылкой на фиг.19-23. После того как результаты выбираемым образом отображены, обработка хоста завершается.

Примеры последовательностей обработки для управления вводом

На фиг.9 показана логическая блок-схема, иллюстрирующая примерный процесс обработки ввода, который выполняется в базовом средстве разработки административных инструментальных средств, показанном на фиг.2. Обработка начинается в блоке 901, где ввод осуществляется посредством программы хоста и направляется к другим компонентам в базовом средстве разработки административных инструментальных средств. Обработка продолжается в блоке 902.

В блоке 902 осуществляется прием ввода из программы хоста. В иллюстративном базовом средстве разработки административных инструментальных средств ввод принимается синтаксическим анализатором, который дешифрирует ввод и направляет ввод для дальнейшей обработки. Обработка продолжается в решающем блоке 904.

В решающем блоке 904 определяется, является ли ввод сценарием. Ввод может принимать форму сценария или строки, представляющей командную строку (далее упоминается как «командная строка»). Командная строка может представлять один или более “cmdlets”, объединенных вместе конвейерной обработкой. Даже если базовое средство разработки административных инструментальных средств поддерживает несколько различных хостов, каждый хост обеспечивает ввод либо сценария, либо командной строки для обработки. Как показано ниже, взаимодействие между сценариями и командными строками является рекурсивным по своей природе. Например, сценарий может иметь строку, которая вызывает “cmdlet”. Собственно “cmdlet” может представлять собой сценарий.

Таким образом, в решающем блоке 904, если ввод имеет форму сценария, обработка продолжается в блоке 906, где выполняется обработка сценария. В противном случае обработка выполняется в блоке 908, где выполняется обработка командной строки. После того как обработка выполнена в любом из блоков 906 или 908, обработка ввода заканчивается.

Пример обработки сценариев

На фиг.10 представлена диаграмма логической последовательности, иллюстрирующая процесс обработки сценария, подходящий для использования в процессе обработки ввода, показанном на фиг.9. Процесс начинается в блоке 1001, где ввод идентифицируется в качестве сенария. Механизм сценария и синтаксический анализатор осуществляют информационный обмен друг с другом для выполнения следующих функций. Обработка продолжается в блоке 1002.

В блоке 1002 над сценарием выполняется предварительная обработка. На фиг.11 показана диаграмма логической последовательности, которая иллюстрирует процесс 1100 предварительной обработки сценария, подходящий для использования в процессе 1000 обработки сценария. Предварительная обработка сценария начинается в блоке 1101 и продолжается в решающем блоке 1102.

В решающем блоке 1102 определяется, начат ли прогон сценария в первый раз. Это определение может основываться на информации, полученной из регистра или другого механизма хранения. Сценарий идентифицируется из механизма хранения, и осуществляется просмотр ассоциированных данных. Если сценарий не исполнялся ранее, то обработка переходит в блок 1104.

В блоке 1104 сценарий регистрируется в регистре. Это позволяет сохранить информацию о сценарии для последующего обращения компонентами в составе базового средства разработки административных инструментальных средств. Обработка продолжается в блоке 1106.

В блоке 1106 из сценария извлекаются справочная информация и документация и сохраняются в регистре. Вновь эта информация может позже использоваться компонентами в составе базового средства разработки административных инструментальных средств. Затем сценарий готов для обработки и возврата в блок 1004 на фиг.10.

В решающем блоке 1102, если при обработке принимается решение, что сценарий исполнялся ранее, обработка переходит к решающему блоку 1108. В решающем блоке 1108 принимается решение, была ли обработка сценария безуспешна. Эта информация может быть получена из регистра. Если для сценария не установлена ошибка, то сценарий готов для обработки и возврата в блок 1004 на фиг.10.

Однако если для сценария установлена ошибка, то обработка продолжается в блоке 1110. В блоке 1110 механизм сценария может уведомить пользователя посредством программы хоста, что для сценария ранее была установлена ошибка. Это уведомление позволит пользователю принять решение, следует продолжить обработку сценария или осуществить выход из сценария. Как упомянуто выше в связи с фиг.8, программа хоста может обрабатывать этот запрос разными путями в зависимости от стиля ввода (например, речевого ввода, командной строки). Как только дополнительный ввод принят от пользователя, сценарий либо возвращается в блок 1004 на фиг.10 для обработки, либо сценарий прерывается.

В блоке 1004 на фиг.10 из сценария извлекается строка. Обработка продолжается в решающем блоке 1006. В решающем блке 1006 принимается решение, включает ли строка в себя ограничения. Ограничение определяется предварительно заданным начальным символом (например, скобкой “[”) и соответствующим конечным символом (например, скобкой “]”). Если строка включает в себя ограничения, то обработка продолжается в блоке 1008.

В блоке 1008 применяются ограничения, включенные в строку. В общем случае ограничения предусматривают механизм в составе базового средства разработки

административных инструментальных средств для определения типа параметра, вводимого в сценарий, и для определения логики подтверждения, которая должна выполняться над параметром. Ограничения применимы не только к параметрам, но также применимы к любому типу логической структуры, вводимой в сценарий, такой как переменные. Таким образом, ограничения обеспечивают механизм в интерпретируемой среде для определения типа данных и для подтверждения параметров. В традиционных средах системные администраторы не имеют возможности формально тестировать параметры, введенные в сценарии. Пример процесса применения ограничений показан на фиг.12.

В решающем блоке 1010 принимается решение, включает ли в себя строка из сценария встроенные характеристики. Встроенные характеристики представляют собой характеристики, которые не выполняются механизмом оболочки. Встроенные характеристики могут обрабатываться с использованием “cmdlets” или могут обрабатываться с использованием других механизмов, таких как подставляемые функции. Если строка не имеет встроенных характеристик, то обработка продолжается в решающем блоке 1014. В противном случае обработка продолжается в блоке 1012.

В блоке 1012 обрабатываются встроенные характеристики, обеспеченные в строке сценария. Примеры встроенных характеристик могут включать в себя исполнение управляющих структур, таких как операторы “if” («если»), циклы “for” («для»), переключатели и т.п. Встроенные характеристики могут также включать в себя операторы типа присваивания (например, a=3). После того как встроенные характеристики обработаны, обработка продолжается в решающем блоке 1014.

В решающем блоке 1014 определяется, включает ли в себя строка сценария командную строку. Определение основано на том, связаны ли данные в строке с командной строкой, которая была зарегистрирована, и с синтаксисом потенциального вызова “cmdlet”. Как упомянуто выше, обработка командных строк и сценариев может быть рекурсивной по характеру, поскольку сценарии могут включать в себя командные строки, и командные строки могут исполнять “cmdlet”, который является собственно сценарием. Если строка не включает в себя командную строку, обработка продолжается в решающем блоке 1018. В противном случае обработка продолжается в блоке 1016.

В блоке 1016 обрабатывается командная строка. В общем представлении обработка командной строки включает в себя идентификацию класса “cmdlet” синтаксическим анализатором и передачу соответствующего объекта “cmdlet” в механизм оболочки для исполнения. Командная строка может также включать в себя конвейерную командную строку, которая подвергается синтаксическому анализу на несколько отдельных объектов “cmdlet” и отдельно обрабатывается механизмом оболочки. Пример процесса обработки командных строк описан ниже со ссылками на фиг.14. Как только командная строка обработана, обработка продолжается в решающем блоке 1018.

В решающем блоке 1018 определяется, имеется ли еще одна строка в сценарии. Если еще одна строка имеется в сценарии, то обработка образует цикл, возвращаясь к блоку 1004, и продолжается в блоках 1004-1016, как описано выше. В противном случае обработка завершается.

Пример процесса применения ограничений в блоке 1008 проиллюстрирован на фиг.12. Процесс начинается в блоке 1201, где в сценарии или в командной строке обнаруживается ограничение. Если ограничение имеется в сценарии, то ограничения и

связанная с ними логическая структура могут возникать на той же самой строке или отдельных строках. Если ограничение находится в командной строке, то ограничение и связанная логическая структура появляются перед индикатором конца строки (например, клавишей “enter”). Обработка продолжается в блоке 1202.

В блоке 1202 ограничения получают из интерпретируемой среды. В иллюстративной среде административных инструментальных средств синтаксический анализатор дешифрирует ввод и определяет появление ограничений. Ограничения могут быть одной из следующих категорий: указатель предиката, указатель синтаксического анализа, указатель подтверждения данных, указатель обработки, указатель кодирования и указатель документирования. В одном примере синтаксического анализа указатели взяты в квадратные скобки и описывают логическую структуру, которая следует за ними. Логическая структура может быть функцией, переменной, сценарием и т.п.

Как описано ниже, за счет использования сценариев разработчики сценариев могут легко вводить и выполнять обработку в отношении параметров в сценарии или командной строке (т.е. интерпретируемой среде), при этом от разработчиков сценариев не требуется генерировать какую-либо лежащую в основе логику. Обработка продолжается в блоке 1204.

В блоке 1204 полученные ограничения сохранятся в метаданных для ассоциированной логической структуры. Ассоциированная логическая структура идентифицируется как представляющая собой первый неатрибутивный маркер после того, как обнаружены один или более атрибутивных маркеров (маркеров, которые обозначают ограничения). Обработка продолжается в блоке 1206.

В блоке 1206, как только логическая структура обнаруживается в сценарии или в командной строке, ограничения, определенные в метаданных, применяются к логической структуре. Ограничения могут включать в себя тип данных, указатели 1210 предиката, указатели 1212 документирования, указатели 1214 синтаксического анализа, указатели 1216 генерации данных, указатели 1218 подтверждения данных и указатели 1220 обработки и кодирования объекта. Ограничения, определяющие типы данных, могут определять любой тип данных, поддерживаемый системой, на которой исполняется базовое средство разработки административных инструментальных средств. Указатели 1210 предиката являются указателями, которые указывают на то, должна ли возникать обработка. Таким образом, указатели 1210 предиката гарантируют, что среда является корректной для исполнения. Например, сценарий может включать в себя следующий указатель предиката:

[PredicateScript(“isInstalled”,“ApplicationZ”].

Указатель предиката гарантирует, что корректное приложение установлено на вычислительном устройстве, перед исполнением сценария. В типовом случае системные переменные среды могут быть определены как указатели предиката. Примеры указателей из типов 1212-1220 указателей иллюстрируются в таблицах 1-5. Затем обработка сценария завершается.

Таким образом, предложенный процесс применения типов и ограничений в интерпретируемой среде позволяет системным администраторам легко определять тип, определять требования подтверждения и т.п., не требуя создания лежащей в основе этого логики для выполнения такой обработки. Ниже приведен пример обработки ограничений, выполняемой над командной строкой, определенной следующим образом:

[Integer][ValidationRange(3,5)]$a=4.

Здесь имеется два ограничения, определенные атрибутивными маркерами, обозначенными “[]”. Первый атрибутивный маркер указывает, что переменная является целочисленным типом, а второй атрибутивный маркер указывает, что значение переменной $a должно быть между 3 и 5 включительно. Данный пример командной строки гарантирует, что если переменная $a присваивается в последующей командной строке, то эта переменная $a будет проверяться на соответствие двум ограничениям. Таким образом, следующие командные строки привели бы к ошибке:

$a=231

$a=”apple”

$a=$(get/location).

Ограничения применяются на различных этапах в рамках базового средства разработки административных инструментальных средств. Например, указатели применимости, указатели документирования и указатели правил синтаксического анализа обрабатываются на самом раннем этапе в синтаксическом анализаторе. Указатели генерирования данных и указатели подтверждения обрабатываются в механизме, как только синтаксический анализатор закончил синтаксический анализ всех параметров ввода.

Следующая таблица иллюстрирует репрезентативные указатели для различных категорий вместе с объяснением обработки, выполняемой средой административных инструментальных средств в ответ на указатель.

Таблица 1
Указатели Применимости

Имя	Описание
Атрибут Требуемой Функции Машины	Информирует оболочку, должен ли элемент использоваться только при определенных функциях машины (например, сервер файлов, почтовый сервер)
Атрибут Требуемой Функции Пользователя	Информирует оболочку, должен ли элемент использоваться только при определенной функции пользователя (например, администратор домена, резервный оператор)
Атрибут Требуемого Сценария	Информирует оболочку, что данный сценарий должен исполняться перед исполнением действительной команды или параметра. Может использоваться для подтверждения параметра
Атрибут Требуемого Типа UI	Используется для проверки имеющегося пользовательского интерфейса перед исполнением

Таблица 2
Указатели Правил Синтаксического Анализа

Имя	Описание
Атрибут Анализа Положения Параметра	Отображает неопределенные параметры на основе положения
Атрибут Анализа Параметра Переменной Длины	Отображает параметры, не имеющие атрибута Анализа Положения Параметра
Атрибут Анализа Запрещения Взаимодействия	Определяет действие, когда число параметров меньше, чем требуемое число
Атрибут Анализа Требования Взаимодействия	Определяет, что параметры получены посредством взаимодействия
Атрибут Анализа Скрытого Элемента	Делает параметр невидимым для конечного пользователя
Атрибут Анализа Обязательного Параметра	Определяет, что параметр является требуемым
Атрибут Анализа Параметра с Паролем	Требует специальной обработки параметра
Атрибут Анализа Строки Подсказки	Определяет подсказку для параметра
Атрибут Анализа Ответа по Умолчанию	Определяет ответ по умолчанию для параметра
Атрибут Анализа Сценария Ответа по Умолчанию	Определяет действие для получения ответа по умолчанию для параметра
Атрибут Анализа Значения по Умолчанию	Определяет значение по умолчанию для параметра
Атрибут Анализа Сценария Значения по Умолчанию	Определяет действие для получения значения по умолчанию для параметра

Атрибут Анализа Отображения Параметров	Определяет способ группирования параметров
Атрибут Анализа Декларации Параметра	Определяет, что поле является параметром
Атрибут Анализа Разрешения Конвейерного ввода	Определяет, что параметр может быть заполнен из конвейерной обработки

Таблица 3
Указатели Документирования

Имя	Описание
Атрибут Документирования Имени	Обеспечивает Имя для ссылки на элементы для взаимодействия или справочной информации
Атрибут Документирования Краткого Описания	Обеспечивает краткое описание элемента
Атрибут Документирования Длинного Описания	Обеспечивает детальное описание элемента
Атрибут Документирования Примера	Обеспечивает пример элемента
Атрибут Документирования «Смотри также»	Обеспечивает список связанных элементов
Атрибут Документирования Конспекта	Обеспечивает информацию документирования для элемента

Таблица 4
Указатели Подтверждения Данных

Имя	Описание
Атрибут Подтверждения Диапазона	Определяет, что параметр должен быть в определенном диапазоне
Атрибут Подтверждения Набора	Определяет, что параметр должен быть в определенной совокупности
Атрибут Подтверждения Шаблона	Определяет, что параметр должен соответствовать некоторому шаблону
Атрибут Подтверждения Длины	Определяет, что строки должны быть в диапазоне размеров
Атрибут Подтверждения Типа	Определяет, что параметр должен быть определенного типа
Атрибут Подтверждения Отсчета	Определяет, что элементов ввода должно быть определенное количество
Атрибут Подтверждения Файла	Определяет некоторые свойства для файла
Атрибут Подтверждения Атрибутов Файла	Определяет некоторые свойства для файла
Атрибут Подтверждения Размера Файла	Определяет, что файлы должны быть в определенном диапазоне
Атрибут Подтверждения Сети	Определяет, что данный Сетевой Объект поддерживает определенные свойства
Атрибут Подтверждения Сценария	Определяет условия оценки перед использованием элемента
Атрибут Подтверждения Метода	Определяет условия оценки перед использованием элемента

Таблица 5
Указатели Обработки и Кодирования

Имя	Описание
Атрибут Обработки Вырезки Строки	Определяет ограничение размера для строк
Атрибут Обработки Вырезки Группы	Определяет ограничение размера для группы
Атрибут Кодирования Приведения Типа	Определяет Тип объектов для кодирования
Атрибут Расширения Шаблонов	Обеспечивает механизм разрешения подстановки

Если рассматриваемое базовое средство разработки административных инструментальных средств работает в базовой структуре .NET™, то каждая категория имеет базовый класс, который выводится из класса базовой категории (например, CmdAttribute). Класс базовой категории выводится из класса системного атрибута System.Attribute. Каждая категория имеет предварительно определенную функцию (например, attrib.func()), которая вызывается синтаксическим анализатором при обработке категории. Разработчик сценария может создать настроенную категорию, которая выведена из класса настроенной категории (например, Cmd.CustomAttribute). Разработчик сценария может также расширить существующий класс категории путем вывода класса указателя из класса базовой категории для данной категории и переопределить предварительно определенную функцию своей реализацией. Разработчик сценария может также переопределить указатели и добавить новые указатели к предварительно определенному набору указателей.

Порядок обработки этих указателей может быть сохранен во внешнем хранилище

данных, доступном для синтаксического анализатора. Базовое средство разработки административных инструментальных средств осуществляет поиск зарегистрированных категорий и вызывает функцию (например, Указатель Настроенной Обработки) для каждого из указателей в этой категории. Таким образом, порядок обработки категорий может быть динамическим за счет сохранения информации об исполнении категрий в постоянной памяти. На различных этапах обработки синтаксический анализатор осуществляет проверку в постоянной памяти, чтобы определить, не требуется ли для какой-либо категории метаданных исполнение в данный момент времени. Это позволяет легко «обесценить» категории путем удаления записи категории из постоянной памяти.

Иллюстративная обработка командной строки

Ниже описан возможный пример обработки командных строк. На фиг.13 показана функциональная схема, графически иллюстрирующая обработку командной строки 1350 посредством синтаксического анализатора 220 и механизма 224 оболочки в составе базового средства разработки административных инструментальных средств, показанного на фиг.2. Примерная командная строка содержит несколько команд (например, команда 1360 обработки, команда 1362 отбора, команда 1364 сортировки, команда 1366 таблицы). Командная строка 1350 может передать параметры ввода в любую из команд (например, “handlecount > 400” передается в команду 1362 отбора). Отметим, что команда 1360 обработки не имеет связанных параметров ввода.

В прошлом каждая команда несла ответственность за синтаксический анализ параметров ввода, связанных с командой, определение того, являются ли действительными параметры ввода, и выдачу сообщений об ошибках, если параметры ввода недействительны. Поскольку команды обычно были написаны разными программистами, синтаксис для параметров ввода в командной строке был не вполне согласованным. Кроме того, если возникала ошибка, то сообщение об ошибке, даже для одной и той же ошибки, было не вполне согласованным для разных команд.

Например, в среде UNIX команда “ls” и команда “ps” имеют ряд несогласованностей между собой. Хотя обе принимают опцию “-w”, опция “-w”, используемая командой “ls”, обозначает ширину страницы, в то время как опция “-w” используется командой “ps” для обозначения вывода ширины печати (по существу, игнорируя ширину страницы). Страницы справочной информации, связанные с командами “ls” и “ps”, также имеют ряд несогласованностей, таких как наличие опций в полужирном начертании в одной из них и отсутствие этого в другой, наличие опций сортировки по алфавиту в одной и отсутствие этого в другой, требование наличия для некоторых опций прерывистых линий и отсутствие этого для других.

Настоящее базовое средство разработки административных инструментальных средств обеспечивает более последовательный подход и минимизирует объем дублированного кода, который должен писать каждый разработчик. Базовое средство 200 разработки административных инструментальных средств обеспечивает синтаксис (например, грамматику), соответствующую семантику (например, словарь) и эталонную модель, чтобы позволить разработчикам легко получать преимущества от общей функциональности, обеспечиваемой базовым средством 200 разработки административных инструментальных средств.

Прежде чем далее описывать настоящее изобретение, ниже приведены определения для дополнительных терминов, встречающихся в этом описании. Термин «параметры ввода» относится к полям ввода для “cmdlet”. Параметр «аргумент» относится к

параметру ввода, передаваемому в “cmdlet”, что эквивалентно одной строке в массиве “argv”, или передаваемому как одиночный элемент в объект “cmdlet”. Как описано ниже, “cmdlet” обеспечивает механизм для определения грамматики. Этот механизм может быть обеспечен прямо или косвенно. Аргумент представляет собой одно из опции, аргумента опции или оператора, за которым следует имя команды. Ниже приведены примеры аргументов, заданные на основе следующей командной строки:

findstr/i/d:\winnt;\winnt\system32 aa*b*.ini.

В приведенной выше командной строке “findstr” является аргументом 0, “/i” - аргументом 1, “/d:\winnt;\winnt\system32” - аргументом 2, “aa*b” - аргументом 3 и “*.ini” - аргументом 4. «Опция» является аргументом для “cmdlet”, который обычно используется для определения изменений в поведении по умолчанию программы. Для приведенного выше примера командной строки “/i” и “/d” являются опциями. «Аргумент опция» является параметром ввода, который следует за определенными опциями. В некоторых случаях «аргумент опция» включен в ту же самую строку аргумента, что и опция. В других случаях «аргумент опция» включен в качестве следующего аргумента. В приведенном выше примере командной строки “winnt; \winnt\system32” является аргументом опции. «Операнд» является аргументом для “cmdlet”, который обычно используется в качестве объекта, предоставляющего в программу информацию, необходимую для завершения обработки программы. Операнды обычно следуют за опциями в командной строке. Для приведенного выше примера “aa*b” и “*.ini” являются операндами. «Синтаксически анализируемый поток» включает в себя аргументы.

В соответствии с фиг.13 синтаксический анализатор 220 подвергает синтаксически анализируемый поток (например, командную строку 1350) анализу по составляющим частям 1320-1326 (например, команде 1322 отбора). Каждая часть 1320-1326 связана с одним из “cmdlets” 1330-1336. Синтаксический анализатор 220 и механизм 224 выполняют различную обработку, такую как синтаксический анализ, подтверждение данных, генерация данных, обработка параметров, кодирование параметров и документирование параметров. Поскольку синтаксический анализатор 220 и механизм 224 выполняют общие функции в отношении параметров ввода в командной строке, то базовое средство 200 разработки административных инструментальных средств имеет возможность выдачи согласованных сообщений об ошибках пользователям.

Можно видеть, что исполняемые “cmdlets” 1330-1336, написанные в соответствии с базовым средством 200 разработки административных инструментальных средств, требуют меньше кода, чем команды в прежних средах администрирования. Каждый исполняемый “cmdlet” 1330-1336 идентифицируется с использованием его соответствующих составных частей 1320-1326. Кроме того, каждый исполняемый “cmdlet” 1330-1336 выводит объекты (представленные стрелками 1340, 1342, 1344 и 1346), которые являются вводом в виде объектов ввода (представленных стрелками 1341, 1343 и 1345) для следующего конвейерного “cmdlet”. Эти объекты могут быть введены путем передачи ссылки (например, дескриптора) к объекту. Исполняемые “cmdlets” 1330-1336 могут затем выполнять дополнительную обработку над объектами, которые были переданы.

На фиг.14 представлена логическая блок-схема, иллюстрирующая более детально обработку командных строк, подходящую для использования в процессе для обработки ввода, показанном на фиг.9. Обработка командной строки начинается в блоке 1401, где синтаксический анализатор или механизм сценария идентифицирует

командную строку во вводе. Обычно механизм оболочки выполняет установку и упорядочивание потока данных “cmdlets”. Ниже описана установка и упорядочение для одного “cmdlet”, однако это применимо к каждому “cmdlet” в конвейерной обработке. Обработка продолжается в блоке 1404.

В блоке 1404 идентифицируется “cmdlet”. Идентификация “cmdlet” может осуществляться через регистрацию. Механизм оболочки определяет, является ли “cmdlet” локальным или удаленным. “cmdlet” может исполняться в следующих местоположениях: 1) в области приложения базового средства разработки административных инструментальных средств; 2) в области другого приложения того же самого процесса, что и в базовом средстве разработки административных инструментальных средств; 3) в другом процессе на том же самом вычислительном устройстве или 4) в удаленном вычислительном устройстве. Информационный обмен между “cmdlets”, действующими в одном и том же процессе, осуществляется через объекты. Информационный обмен между “cmdlets”, действующими в разных процессах, осуществляется через формат структурированных данных, преобразованный в последовательную форму. Иллюстративный формат структурированных данных, преобразованный в последовательную форму, онован на языке XML. Обработка продолжается в блоке 1406.

В блоке 1406 создается объект “cmdlet”. Пример процесса создания экземпляра “cmdlet” описан ниже со ссылкой на фиг.15. Как только объект “cmdlet” создан, обработка продолжается в блоке 1408.

В блоке 1408 осуществляется заполнение свойств, связанных с объектом “cmdlet”. Как описано выше, разработчик декларирует свойства в классе “cmdlet” или во внешнем источнике. Описывая кратко, базовое средство разработки административных инструментальных средств будет дешифрировать входящие объекты для “cmdlet”, реализованного из класса “cmdlet” на основе имени и типа, которые декларированы для свойства. Если типы различаются, то может быть осуществлено приведение типа посредством администратора расширенного типа данных. Как упомянуто выше, в случае конвейерных командных строк выходным результатом каждого “cmdlet” может быть список дескрипторов для объектов. Следующий “cmdlet” вводит этот список дескрипторов объектов, выполняет обработку и передает другой список дескрипторов объектов в следующий “cmdlet”. Кроме того, как показано на фиг.7, параметры ввода могут быть определены как поступающие из командной строки. Иллюстративный способ заполнения свойств, связанных с “cmdlet”, описан ниже со ссылкой на фиг.16. Как только “cmdlet” заполнен, обработка продолжается в блоке 1410.

В блоке 1410 происходит выполнение “cmdlet”. В общем представлении обработка, обеспечиваемая “cmdlet”, выполняется, по меньшей мере, однократно, что включает в себя обработку для каждого ввода объекта в “cmdlet”. Таким образом, если “cmdlet” является первым “cmdlet” в конвейерной командной строке, то обработка выполняется однократно. Для последующих “cmdlets” обработка выполняется для каждого объекта, который передается в “cmdlet”. Иллюстративный способ исполнения “cmdlets” описан ниже со ссылкой на фиг.5. Когда параметры ввода представляют собой только такие, которые поступают из командной строки, исполнение “cmdlet” использует методы, установленные по умолчанию, обеспечиваемые базовым случаем “cmdlet”. После завершения исполнения “cmdlet” обработка переходит к блоку 1412.

В блоке 1412 “cmdlet” завершается (возвращает ресурсы системе). Это включает вызов элемента разрушения для соответствующего объекта “cmdlet”, ответственного

за отмену распределения памяти и т.п. Затем обработка командной строки завершается.

Иллюстративный процесс создания объекта “cmdlet”

На фиг.15 показана логическая блок-схема, иллюстрирующая пример процесса создания объекта “cmdlet”, подходящего для использования в обработке командных строк, показанной на фиг.14. В этот момент структура данных “cmdlet” разработана, и атрибуты и ожидаемые параметры ввода определены. “cmdlet” скомпилирован и зарегистрирован. В процессе регистрации имя класса (т.е. имя “cmdlet”) записано в памяти регистрации, и метаданные, связанные с “cmdlet”, сохранены. Процесс 1500 начинается в блоке 1501, где синтаксический анализатор получает ввод (например, нажатия клавиш), указывающий на “cmdlet”. Синтаксический анализатор может распознать ввод как “cmdlet” путем поиска ввода в регистре и ассоциирования ввода с одним из зарегистрированных “cmdlets”. Затем обработка переходит к блоку 1504.

В блоке 1504 метаданные, связанные с классом объекта “cmdlet”, считываются. Метаданные включают в себя любые из указателей, связанных с “cmdlet”. Указатели могут применяться к собственно “cmdlet” или к одному или более параметрам. При регистрации “cmdlet” код регистрации регистрирует метаданные в постоянной памяти. Метаданные могут быть сохранены в XML-файле в последовательном формате, во внешней базе данных и т.п. Подобно обработке указателей при обработке сценария, каждая категория указателей обрабатывается на отдельном этапе. Каждый указатель метаданных определяет свою собственную обработку ошибок. Затем обработка продолжается в блоке 1506.

В блоке 1506 объект “cmdlet” реализуется на основе идентифицированного класса “cmdlet”. Обработка продолжается в блоке 1508.

В блоке 1508 обеспечивается информация о “cmdlet”. Это может осуществляться посредством отражения или иным образом. Эта информация относится к ожидаемым параметрам ввода. Как отмечено выше, параметры, которые декларируются общедоступным способом (например, общедоступное Имя 730 строки), соответствуют ожидаемым параметра ввода, которые могут быть определены в командной последовательности в командной строке или обеспечены в потоке ввода. Базовое средство разработки административных инструментальных средств через администратора расширенного типа, представленного на фиг.18, обеспечивает общий интерфейс для возврата информации (по мере необходимости) к вызывающему оператору. Обработка продолжается в блоке 1510.

В блоке 1510 применяются указатели применимости (например, Таблица 1). Указатели применимости гарантируют, что класс используется в некоторых машинных функциях и/или в пользовательских функциях. Например, некоторые “cmdlets” могут быть использованы только Администраторами Доменов. Если ограничение, определенное в одном из указателей применимости, не удовлетворяется, то возникает ошибка. Обработка продолжается в блоке 1512.

В блоке 1512 метаданные используются для обеспечения интеллектуального восприятия. В этот момент обработки вся командная строка еще не введена. Однако в базовом средстве разработки административных инструментальных средств известны применимые “cmdlets”. Как только “cmdlet” определен, базовому средству разработки административных инструментальных средств становятся известными параметры ввода, которые разрешены, за счет отражения на объект “cmdlet”. Таким образом, базовое средство разработки административных инструментальных средств может автоматически завершать “cmdlet”, как только устраняющая неоднозначность часть

имени “cmdlet” обеспечена, и затем автоматически завершать параметр ввода, как только устраняющая неоднозначность часть параметра ввода напечатана на командной строке. Автоматическое завершение может возникать сразу же, как только часть параметра ввода может однозначным образом идентифицировать один из параметров ввода. Кроме того, автоматическое завершение может выполняться также для имен и операндов “cmdlet”. Затем обработка продолжается в блоке 1514.

В блоке 1514 обработка ожидает до тех пор, пока параметры ввода для “cmdlet” не будут введены. Это может произойти после того, как пользователь показал конец командной последовательности, например, путем нажатия клавиши ввода. В сценарии новая строка указывает конец командной последовательности. Это ожидание может включать получение дополнительной информации от пользователя, касающейся параметров и применения других указателей. Если “cmdlet” является одним из конвейерных параметров, обработка может начаться немедленно. После того как необходимая командная последовательность и параметры ввода обеспечены, обработка завершается.

Иллюстративный процесс заполнения “cmdlet”

Пример процесса заполнения “cmdlet” показан на фиг.16 и описан ниже во взаимосвязи с фиг.5. В одном варианте базового средства разработки административных инструментальных средств механизм оболочки выполняет обработку для заполнения параметров для “cmdlet”. Обработка начинается в блоке 1601 после того, как экземпляр базовое средство разработки административных инструментальных средств создан. Затем обработка продолжается в блоке 1602.

В блоке 1602 извлекается параметр (например, Имя Процесса), декларированный в “cmdlet”. На основе декларации в “cmdlet” механизм оболочки распознает, что входящие объекты ввода будут обеспечивать свойство, именованное «Имя Процесса». Если тип входящего свойства отличается от типа, определенного в декларации параметра, будет осуществлено приведение типа с помощью администратора расширенного типа. Процесс приведения типов данных пояснен ниже в подразделе, озаглавленном «Пример обработки администратора расширенного типа». Затем обработка продолжается в блоке 1603.

В блоке 1603 получают атрибут, связанный с параметром. Атрибут идентифицирует, является ли источник ввода для параметра командной линией или он исходит из конвейерной обработки. Затем обработка переходит к решающему блоку 1604.

В решающем блоке 1604 принимается решение, определяет ли атрибут источник ввода как командную строку. Если источник ввода является командной строкой, то обработка продолжается в блоке 1609. В противном случае обработка продолжается в решающем блоке 1605.

В решающем блоке 1605 принимается решение, должно ли использоваться имя свойства, определенное в декларации, или должно использоваться отображение для имени свойства. Это определение основывается на том, определял ли ввод команды отображение для параметра. Следующая строка иллюстрирует пример отображения параметра «Имя Процесса» на элемент “foo” («что угодно») входящего объекта.

$ get/process|where han* -gt 500|stop/procress-ProcessName<-foo.

Обработка продолжается в блоке 1606.

В блоке 1606 применяется отображение. Отображение заменяет имя ожидаемого параметра с “ProcessName” на “foo”, которое затем используется механизмом оболочки для синтаксического анализа входящих объектов и для идентификации

корректного ожидаемого параметра. Затем обработка продолжается в блоке 1608.

В блоке 1608 администратор расширенного типа запрашивается для определения положения значения для параметра во входящем объекте. Как пояснено в связи с администратором расширенного типа, администратор расширенного типа берет имя параметра и использует отражение для идентификации параметра во входящем объекте с именем параметра. Администратор расширенного типа может также, при необходимости, выполнять другую обработку для параметра. Например, администратор расширенного типа может осуществлять приведение типа данных к ожидаемому типу данных с помощью механизма преобразования, описанного выше. Затем обработка продолжается в решающем блоке 1610.

Возвращаясь к блоку 1609, если атрибут определяет, что входной источник является командной строкой, то данные получают из командной строки. Получение данных из командной строки может выполняться посредством администратора расширенного типа. Затем обработка переходит к решающему блоку 1610.

В решающем блоке 1610 определяется, имеется ли еще другой ожидаемый параметр. Если другой ожидаемый параметр имеется, то обработка возвращается к блоку 1602 и продолжается, как описано выше. В противном случае обработка завершается и осуществляется выход.

Таким образом, как показано выше, “cmdlet” действует как шаблон для «измельчения» входных данных для получения ожидаемых параметров. Кроме того, ожидаемые параметры получаются без знания типа входящего объекта, обеспечивающего значение для ожидаемого параметра. Это полностью отличается от традиционных сред администрирования. Традиционные среды администрирования являются сильно связанными и требуют, чтобы тип объекта был известен к моменту компилирования. Кроме того, в традиционных средах ожидаемый параметр должен был быть введен в функцию по значению или по ссылке. Таким образом, предложенный механизм синтаксического анализа (например, «измельчение») позволяет программистам определять тип параметра, не требуя конкретного знания, каким образом получены значения для этих параметров.

Например, при задании следующей декларации для “cmdlet” “Foo”:

```
class Foo : Cmdlet
{
    string Name;
    Bool Recurse;
}
```

Синтаксис командной строки может быть любым из следующего:

$ Foo-Name: (string)-Recurse:True

$ Foo-Name <string>-Recurse True

$ Foo-Name (string).

Набор правил может быть модифицирован системными администраторами, чтобы получить желательный синтаксис. Кроме того, синтаксический анализатор может поддерживать множество наборов правил, так что пользователями может использоваться более одного синтаксиса. По существу, грамматика, связанная со структурой “cmdlet” (например, string Name (Имя строки) и Bool Recurse (булева

рекурсия)), управляет синтаксическим анализатором.

В принципе, указатели синтаксического анализа описывают, каким образом параметры, введенные как командная последовательность, должны отображаться на ожидаемые параметры, идентифицированные в объекте “cmdlet”. Типы параметров ввода проверяются для определения корректности. Если типы параметров ввода не корректны, то для параметров ввода может быть осуществлено приведение для обеспечения их корректности. Если типы параметров ввода не корректны и не могут быть приведены, то выводится ошибка использования. Ошибка использования позволяет пользователю получить информацию о том, что ожидается корректный синтаксис. Ошибка использования может получить информацию, описывающую синтаксис из Указателей Документирования. После того как типы параметров ввода либо отображены, либо проверены, соответствующие элементы в экземпляре объекта “cmdlet” заполняются. По мере того как элементы заполняются, администратор расширенного типа обеспечивает обработку типов параметров ввода. Описывая кратко, обработка может включать в себя механизм пути свойства, механизм ключа, еханизм сравнения, механизм преобразования, механизм подстановки, механизм отношения и механизм установки свойств. Каждый из этих механизмов подробно описан ниже в разделе, озаглавленном «Обработка администратора расширенного типа», в который также включены иллюстративные примеры.

Иллюстративный процесс для исполнения “cmdlet”

Иллюстративный процесс для исполнения “cmdlet” представлен на фиг.17 и описан ниже. В иллюстративной среде административных инструментальных средств механизм оболочки исполняет “cmdlet”. Как упомянуто выше, код 1442 во втором методе 1440 исполняется для каждого объекта ввода. Обработка начинается в блоке 1701, где “cmdlet” уже заполнен. Затем обработка продолжается в блоке 1702.

В блоке 1702 оператор 542 из кода 542 извлекается для исполнения. Обработка продолжается в решающем блоке 1704.

В решающем блоке 1704 принимается решение, содержит ли оператор «зацепку» (подключение программы). Со ссылкой на фиг.5 такое подключение программы может включать в себя вызов API, обеспечиваемого механизмом оболочки. Например, оператор 550 в коде 542 “cmdlet” 540 на фиг.5 вызывает confirmprocessing API (API обработки подтверждения), определяющий необходимые параметры, первую последовательность (например, “PID=”) и параметр (например, PID). Вновь в соответствии с фиг.17, если оператор включает в себя подключение программы, обработка продолжается в блоке 1712. Таким образом, если определена команда, вызывающая API обработки подтверждения, то “cmdlet” действует в альтернативном режиме исполнения, который обеспечивается операционной средой. В противном случае обработка продолжается в блоке 1706, и исполнение продолжается в «нормальном» режиме.

В блоке 1706 обрабатывается оператор. Обработка затем переходит к решающему блоку 1708. В блоке 1708 определяется, включает ли в себя код другой оператор. Если имеется другой оператор, то обработка возвращается к блоку 1702 для получения следующего оператора и обработка продолжается, как описано выше. В противном случае обработка продолжается в решающем блоке 1714.

В решающем блоке 1714 определяется, имеется ли другой объект ввода для обработки. Если имеется другой объект ввода для обработки, то обработка продолжается в блоке 1716, где “cmdlet” заполняется данными из следующего объекта. Процесс заполнения, описанный со ссылками на фиг.16, выполняется в отношении

следующего объекта. Обработка затем возвращается к блоку 1702 и продолжается, как описано выше. После того как все объекты обработаны, процесс исполнения “cmdlet” завершается и осуществляется выход.

Возвращаясь к решающему блоку 1704, если оператор включает в себя подключение программы, обработка продолжается в блоке 1712. В блоке 1712 обрабатываются дополнительные свойства, обеспечиваемые средой административных инструментальных средств. Обработка переходит к решающему блоку 1708 и продолжается, как описано выше.

Дополнительная обработка, выполняемая в блоке 1712, описана далее во взаимосвязи с примерной структурой 600 данных, показанной на фиг.6. Как пояснено выше, в базовом классе 600 команд могут быть декларированные параметры, которые соответствуют дополнительным ожидаемым параметрам ввода (например, переключатель).

Переключатель включает предварительно определенную последовательность и, при распознавании, предписывает механизму оболочки обеспечить дополнительную функцию для “cmdlet”. Если словесное описание 610 параметра определено в командном вводе, то исполняются словесные операторы 614. Ниже приведен пример командной строки, которая включает в себя переключатель словесного определения:

$ get/process|where “han*-gt500”|stop/process-verbose.

В принципе, когда в командном вводе определено “-verbose” (словесный), механизм оболочки исполняет команду для каждого объекта ввода и направляет действительную команду, которая исполнялась для каждого объекта ввода, в программу хоста для отображения. Ниже приведен пример вывода, генерируемого, когда вышеуказанная командная строка исполняется в иллюстративной среде административных инструментальных средств:

$ stop/process PID=15

$ stop/process PID=33.

Если в командном вводе определен параметр “whatif” («что если») 620, то исполняются операторы “whatif” («что если») 624. Ниже приведен пример командной строки, которая включает в себя переключатель “whatif”:

$ get/process|where “han*-gt500”|stop/process-whatif.

В принципе, если определено “-whatif”, механизм оболочки не исполняет в действительности код 542, а вместо этого посылает команды, которые должны были исполняться, в программу хоста для отображения. Ниже приведен пример вывода, генерируемого, когда приведенная выше командная строка исполняется в среде административных инструментальных средств согласно настоящему изобретению:

#$ stop/process PID=15

#$ stop/process PID=33.

Если в командном вводе определен параметр “confirm” («подтвердить») 630, то исполняются операторы “confirm” 634. Ниже приведен пример командной строки, которая включает в себя переключатель “confirm”:

$ get/process|where “han*-gt500”|stop/process-confirm.

В принципе, если определено “-confirm”, механизм оболочки запрашивает дополнительный пользовательский ввод относительно того, продолжать команду или нет. Ниже приведен пример вывода, генерируемого, когда приведенная выше командная строка исполняется в среде административных инструментальных средств согласно настоящему изобретению:

$ stop/process PID 15

Y/N Y
$ stop/process PID 33
Y/N N.

Как описано выше, пример структуры данных 600 может также включать в себя метод 640 защиты, который определяет, должна ли быть разрешена задача, запрашиваемая для исполнения. В традиционных средах администрирования каждая команда ответственна за проверку того, имеет ли сторона, исполняющая команду, достаточно привилегий для выполнения команды. Для выполнения этой проверки необходим расширенный код для доступа к информации от различных источников. Ввиду этих сложностей многие команды не выполняют проверку защиты. Авторы заявленной среды административных средств исходили из того, что, когда задача определена в командном вводе, необходимая информация для выполнения проверки защиты имеется в среде административных инструментальных средств. Поэтому базовое средство разработки административных инструментальных средств выполняет проверку защиты, не требуя сложного кода от разработчиков инструментальных средств. Проверка защиты может быть выполнена для любого “cmdlet”, который определяет подключение программы для данного “cmdlet”. Альтернативно, подключение программы может быть факультативным параметром ввода, который может быть определен в командном вводе, подобно словесному параметру, описанному выше.

Проверка защиты реализуется для поддержки аутентификации на основе функций, что в общем случае определяется как система контроля, какие пользователи имеют доступ к ресурсам на основе функции пользователя. Таким образом, каждой функции присваиваются определенные права доступа к различным ресурсам. Пользователю затем присваиваются одна или несколько функций. В общем случае аутентификация, основанная на функциях, фокусируется на трех элементах: принцип, ресурс и действие. Принцип идентифицирует того, кто запросил действие, подлежащее выполнению над ресурсом.

Авторы настоящего изобретения исходили из того, что запрашиваемый “cmdlet” соответствует действию, которое должно выполняться. Кроме того, учитывалось то, что владелец процесса, в котором исполняется базовое средство разработки административных инструментальных средств, соответствует главному пользователю. Кроме того, изобретатели имели в виду, что ресурс определен в “cmdlet”. Поэтому, поскольку базовое средство разработки административных инструментальных средств имеет доступ к этим элементам, изобретатели предусматривали, что проверка защиты могла бы выполняться из базового средства разработки административных инструментальных средств, не требуя от разработчиков реализовывать проверку защиты.

Операция проверки защиты может быть выполнена каждый раз, когда дополнительная функция запрашивается в “cmdlet” с использованием подключения программы, например, confirmprocessing API. Альтернативно проверка защиты может выполняться путем проверки, был ли введен переключатель защиты в командную строку, подобно verbose, whatif и confirm. Для любой реализации метод проверки защиты вызывает API, обеспечиваемый процессом защиты (не показано), который обеспечивает набор API для определения того, кто имеет разрешение. Процесс защиты получает информацию, обеспечиваемую базовым средством разработки административных инструментальных средств, и обеспечивает результат, указывающий, может ли задача быть завершена. Базовое средство разработки

административных инструментальных средств может затем сформировать сообщение об ошибке или остановить выполнение задачи.

Таким образом, путем обеспечения подключения программы в “cmdlet” разработчики могут использовать дополнительную обработку, обеспечиваемую базовым средством разработки административных инструментальных средств.

Пример обработки администратора расширенного типа

Как кратко упомянуто выше в связи с фиг.18, администратор расширенного типа может выполнить дополнительную обработку на объектах, которые предоставлены. Дополнительная обработка может быть выполнена по запросу синтаксического анализатора 220, механизма 222 сценария или процессора 402 конвейерной обработки. Дополнительная обработка включает в себя механизм пути свойств, механизм ключа, механизм сравнения, механизм преобразования, механизм подстановки, механизм отношения и механизм установки свойств. Специалистам в данной области техники должно быть понятно, что администратор расширенного типа может также быть расширен в отношении другой обработки без отклонения от объема изобретения. Ниже описан каждый из дополнительных механизмов обработки.

Во-первых, механизм пути свойств позволяет строке просмотреть свойства объектов. В современных системах отражения запросы могут запрашивать свойства объекта. Однако в предложенном администраторе расширенного типа может быть определена строка, которая обеспечить путь просмотра последовательных свойств объектов. Ниже приведен иллюстративный синтаксис для пути свойств: P1.P2.P3.P4. Каждый компонент (например, P1, P2, P3 и P4) содержит строку, которая может представлять свойство, метод с параметрами, метод без параметров, поле, XPATH и т.п. XPATH определяет строку запроса на поиск элемента (например, “/FOO@=13”). В пределах строки специальный символ может быть включен для конкретного указания типа компонента. Если строка не содержит специального символа, администратор расширенного типа может выполнить поиск (просмотр) для определения типа компонента. Например, если компонент P1 является объектом, администратор расширенного типа может запросить, является ли P2 свойством объекта, методом для объекта, полем объекта или набором свойств. После того как администратор расширенного типа идентифицирует тип для P2, выполняется обработка, соответствующая этому типу. Если компонент не является ни одним из указанных выше типов, то администратор расширенного типа может дополнительно запросить расширенные источники для определения, имеется ли функция преобразования для преобразования типа P1 в тип P2. Эти и другие поиски описаны ниже с использованием иллюстративных командных строк и представлением соответствующих выходных результатов.

Ниже представлена иллюстративная строка, которая включает в себя путь свойств:

$ get/process|where hand*-gt>500|format/table name.toupper, ws.kb, exe*.ver*.description.tolower.trunc(30)

В приведенной выше иллюстративной строке имеются три пути свойств: (1) “name.toupper”; (2) “ws.kb” и (3) “exe*.ver*. description.tolower.trunc(30)”. Перед описанием этих путей свойств следует отметить, что “name”, “ws” и “exe” определяют свойства для таблицы. Кроме того, следует отметить, что каждое из этих свойств является прямым свойством входящего объекта, исходно генерированным “get/process” и затем переданным в конвейерной обработке через различные “cmdlets”. Ниже описана обработка для каждого из этих трех путей свойств.

В первом пути свойства (т.е. “name.toupper”) имя является прямым свойством

входящего объекта и само также является объектом.

Администратор расширенного вызова запрашивает систему с использованием приоритетного поиска, описанного выше, для определения типа для “toupper”. Администратор расширенного типа обнаруживает, что “toupper” не является свойством. Однако “toupper” может быть методом, унаследованным типом строки для преобразования букв нижнего регистра в буквы верхнего регистра в строке. Альтернативно, администратор расширенного типа может запрашивать расширенные метаданные для определения того, имеется ли какой-либо код третьей стороны, который может преобразовать объект имени в верхний регистр. После нахождения типа компонента обработка выполняется в соответствии с этим типом компонента.

Во втором пути (т.е. “ws.kb”) “ws” является прямым свойством входящего объекта и также собственно объектом. Администратор расширенного типа определяет, что “ws” является целым числом. Затем администратор расширенного типа запрашивает, является ли “kb” свойством целого числа, является ли “kb” методом для целого числа, и, наконец, запрашивает, знает ли какой-либо код, как взять целое число и преобразовать целое число в тип “kb”. Код третьей стороны зарегистрирован для выполнения этого преобразования, и преобразование выполняется.

В третьем пути свойства (т.е. “exe*.ver*. description.tolower.trunc(30)”) имеется несколько компонентов. Первый компонент (“exe*”) является прямым свойством входящего объекта и также является объектом. Вновь администратор расширенного типа переходит к запросу просмотра для обработки второго компонента (“ver*”). Объект “exe*” не имеет свойства или метода “ver*”, так что администратор расширенного типа запрашивает расширенные метаданные для определения того, имеется ли какой-либо код, который зарегистрирован для преобразования исполняемого имени в версию. В этом примере такой код существует. Этот код может взять исполняемую строку имени и использовать ее для открытия файла, затем обращается к объекту блока версии и возвращает свойство описания (третий компонент “description”) для объекта блока версии. Администратор расширенного типа затем выполняет тот же самый механизм поиска для четвертого компонента (“tolower”) и пятого компонента (“trunk(40)”). Таким образом, как показано, администратор расширенного типа может выполнить весьма сложную обработку над командной строкой, при этом администратору не требуется создавать какой-либо специфический код. Таблица 1 иллюстрирует выходной результат, генерируемый для рассмотренной последовательности.

Таблица 1

Name.toupperws.kbexe*.ver*.description.tolower.trunc(30)
ETCLIENT 29,964 etclient
CSRSS 6,944
SVCHOST 28,944 generic host process for win32
OUTLOOCK 18,556 office outloock
MSMSGS 13,248 messenger

Другой механизм 1824 запроса включает в себя ключ. Ключ идентифицирует одно или более свойств, которые могут сделать экземпляр типа данных уникальным. Например, в базе данных один столбец может быть идентифицирован как ключ, который может уникальным образом идентифицировать каждую строку (например, номер социальной защиты). Ключ хранится в метаданных 1840 типа, связанных с типом данных. Этот ключ может затем использоваться администратором расширенного типа при обработке объектов этого типа данных. Тип данных может

быть расширенным типом данных или существующим типом данных.

Еще один механизм запроса может включать в себя механизм сравнения. Механизм сравнения сравнивает два объекта. Если два объекта непосредственно поддерживают функцию сравнения, то исполняется непосредственно поддерживаемая функция сравнения. Однако если ни один объект не поддерживает функцию сравнения, то администратор расширенного типа может осуществить поиск в метаданных типа для нахождения кода, который был зарегистрирован для поддержки функции сравнения между двумя объектами. Иллюстративная последовательность командных строк, вызывающих механизм сравнения, показана ниже, вместе с соответствующим выходным результатом, в Таблице 2.

Таблица 2

```
$ $a=(get/date)
$ start/sleep 5
$ $b=$(get/date
compare/time $a $b

Ticks                :51196579
Days                 :0
Hours                :0
Milliseconds         :119
Minutes              :0
Seconds              :5
TotalDays            :5.92552997685185E-05
TotalHours           :0.00142212719444444
TotalMilliseconds    :5119.6579
TotalMinutes         :0.0853276316666667
TotalSeconds         :5.1196579
```

“cmdlet” “compare/time” («сравнить/время») написан для сравнения двух объектов DataTime («дата/время»). В этом случае объект DataTime поддерживает интерфейс IComparable («сравнимое»).

Еще один механизм 1824 запроса включает в себя механизм преобразования. Администратор расширенного типа позволяет регистрировать код с указанием его возможности выполнять конкретное преобразование. Тогда, если объект типа A вводится и “cmdlet” определяет объект типа B, то администратор расширенного типа может выполнить преобразование с использованием зарегистрированных преобразований. Администратор расширенного типа может выполнить ряд преобразований для приведения типа A к типу B. Путь свойств, описанный выше (“ws.kb”), иллюстрирует механизм преобразования.

Другой механизм 1824 преобразования включает в себя механизм подстановки. Подстановка относится к универсальному символу (знаку подстановки) в строке. Механизм подстановки вводит строку со знаком подстановки и формирует набор объектов. Администратор расширенного типа позволяет регистрировать код, который определяет обработку знака подстановки. Путь свойства, описанный выше (“exe*.ver*. description.tolower.trunc(30)”), иллюстрирует механизм подстановки. Зарегистрированный процесс может обеспечивать подстановку для имен файлов, объектов файлов, входящих свойств и т.п.

Другой механизм 1824 запроса включает в себя механизм установки свойств. Механизм установки свойств позволяет определить имя для набора свойств. Администратор может затем определить имя в командной строке для получения

набора свойств. Набор свойств может быть определен различным образом. В одном способе предварительно определенный параметр, такой как “?”, может быть введен в качестве параметра ввода для “cmdlet”. Операционная среда после распознавания предварительно определенного параметра перечисляет все свойства входящего объекта. Список может представлять собой графический пользовательский интерфейс GUI, который позволяет администратору легко проверять (например, выбором мышью) желательные свойства и имя набора свойств. Информация набора свойств затем сохраняется в расширенных метаданных. Иллюстративная строка, вызывающая механизм установки свойств, показана ниже, вместе с соответствующим выходным результатом, в Таблице 3:

$ get/process|where han*-gt>500|format/table config.

В этой иллюстративной строке набор свойств, именованный как “config”, определен для включения свойства имени, свойства идентификатора процесса (Pid) и свойства приоритета. Выходной результат для таблицы показан ниже.

Таблица 3

Name	PidPriority
ETClient	3528 Normal
csrss	528 Normal
svchost	848 Normal
OUTLOOK	2,772 Normal
msmsgs	2,584 Normal

Другой механизм 1824 запроса включает в себя механизм отношения. В противоположность системам традиционного типа, которые поддерживают одно отношение (т.е. наследование), механизм отношения поддерживает более одного соотношения между типами. Вновь эти отношения регистрируются. Отношение сможет включать в себя нахождение элементов, которые объект использует, или нахождение элементов, которые используют объект. Администратор расширенного типа может обращаться к онтологиям, которые описывают различные отношения. Используя расширенные метаданные и код, может быть описана спецификация для обращения к любому сервису онтологии, такому как OWL, DAWL и т.п. Примером части иллюстративной строки, которая использует механизм отношения, может быть следующая: .OWL:”string”.

Идентификатор “OWL” идентифицирует сервис онтологии, а “string” определяет конкретную строку в сервисе онтологии. Таким образом, администратор расширенного типа может обращаться к типам, выдаваемым сервисами онтологии.

Пример процесса отображения данных командной строки

Предложенный механизм обеспечивает управляемый данными вывод командной строки. Форматирование и вывод данных обеспечиваются одним или более “cmdlets” в конвейере “cmdlets”. В типовом случае “cmdlets” включены в неуправляющие “cmdlets”, описанные со ссылкой на фиг.2. “cmdlets” могут включать в себя “cmdlet” форматирования, “cmdlet” разметки, “cmdlet” преобразования, “cmdlet” трансформации и “cmdlet” вывода.

На фиг.19 графически представлены последовательности 1901-1907 этих “cmdlets” в конвейерной обработке. Первая последовательность 1901 иллюстрирует “cmdlet” 1910 вывода как последний “cmdlet” в конвейерной обработке. Тм же способом, как описано выше для других “cmdlets”, “cmdlet” 1910 вывода получает поток объектов конвейерной обработки, генерированных и обработанных другими “cmdlets” в конвейерной обработке. Однако в отличие от большинства “cmdlets” “cmdlet” 1910

вывода не создает объектов конвейерной обработки для других “cmdlets”. Вместо этого “cmdlet” 1910 вывода обеспечивает визуализацию/отображение результатов конвейерной обработки. Каждый “cmdlet” 1910 вывода связан с местом назначения вывода, таким как устройство, программа и т.п. Например, для устройства консоли “cmdlet” 1910 вывода может быть определен как “out/console”; для Интернет-браузера “cmdlet” 1910 вывода может быть определен как “out/browser”; для окна “cmdlet” 1910 вывода может быть определен как “out/window”. Каждый конкретный “cmdlet” вывода знаком со свойствами связанного с ним места назначения. Локальная информация (например, дата и формат представления валюты) обрабатывается “cmdlet” 1910 вывода, если только “cmdlet” преобразования не предшествует “cmdlet” вывода в конвейерной обработке. В этой ситуации локальную информацию преобразует “cmdlet” преобразования.

Каждый хост ответственен за поддержку определенных “cmdlet” вывода, таких как “out/console”. Хост также поддерживает любой специфический для места назначения “cmdlet” хоста (например, “out/chart”, который направляет вывод в график, обеспечиваемый приложением электронной таблицы). Кроме того, хост несет ответственность за обеспечение обработки по умолчанию результатов. “cmdlet” вывода в этой последовательности может принять решение реализовать свое поведение путем вызова других “cmdlets” обработки вывода (таких, как форматирования, разметки, преобразования и трансформации). Таким образом, “cmdlet” вывода может неявным образом модифицировать последовательность 1901 в любую из других последовательностей или может добавить свои собственные дополнительные “cmdlets” форматирования/вывода.

Вторая последовательность 1902 иллюстрирует “cmdlet” 1920 форматирования перед “cmdlet” 1910 вывода. В случае этой последовательности “cmdlet” 1920 форматирования получает поток объектов конвейерной обработки, генерированных и обработанных другими “cmdlets” в конвейерной обработке. В общем представлении “cmdlet” 1920 форматирования обеспечивает способ выбора свойств отображения и способ определения стиля страницы, включая форму, ширину столбцов, заголовки, нижние колонтитулы и т.п. Форма может включать в себя таблицу, список широкого типа, многоколоночный список и т.п. Кроме того, “cmdlet” 1920 формата может включать в себя вычисления итогов или сумм. Примерная обработка, выполняемая “cmdlet” 1920 форматирования, описана ниже со ссылкой на фиг.20. Описывая кратко, “cmdlet” форматирования выдает объекты форматирования в дополнение к выдаче объектов конвейерной обработки. Объекты форматирования могут быть распознаны ниже по потоку обработки посредством “cmdlet” вывода (например, “cmdlet” 1920 вывода в последовательности 1902) с помощью администратора расширенного типа или иного механизма. “cmdlet” 1920 вывода может выбрать либо использование выданных объектов форматирования, либо их игнорирование. В некоторых случаях модификации в стиле страницы могут определяться “cmdlet” вывода. В примерном процессе “cmdlet” вывода может установить неопределенную ширину столбцов путем нахождения максимальной длины для каждого свойства из предварительно заданного числа объектов (например, 50) и установить ширину столбца на максимальную длину. Объекты форматирования включают в себя информацию форматирования, информацию заголовка, нижнего колонтитула и т.п.

Третья последовательность 1903 иллюстрирует “cmdlet” 1920 форматирования перед “cmdlet” 1910 вывода. Однако в третьей последовательности 1903 “cmdlet” 1930 разметки введен в конвейерную обработку между “cmdlet” 1920 форматирования и

“cmdlet”1910 вывода. “cmdlet” 1930 разметки обеспечивает механизм для добавления аннотации свойств (например, шрифт, цвет) для выбранных параметров. Таким образом, “cmdlet” 1930 разметки возникает перед “cmdlet” 1910 вывода. Аннотации свойств могут быть реализованы с использованием «дублирующего множества свойств» или путем добавления аннотаций свойств в настроенном пространстве имен во множестве свойств. “cmdlet” 1930 разметки может возникать перед “cmdlet” 1920 форматирования, если аннотации разметки могут поддерживаться при обработке “cmdlet” 1920 форматирования.

Четвертая последовательность 1904 вновь иллюстрирует “cmdlet” 1920 форматирования перед “cmdlet” 1910 вывода. Однако в четвертой последовательности 1904 между “cmdlet” 1920 форматирования и “cmdlet” 1910 вывода в конвейерную обработку введен “cmdlet” 1940 преобразования. “cmdlet” 1940 преобразования конфигурирован, таким образом, для обработки объектов форматирования, выданных “cmdlet” 1920 форматирования. “cmdlet” 1940 преобразования преобразует объекты конвейерной обработки в конкретное кодирование на основе объектов форматирования. “cmdlet” 1940 преобразования связан с конкретным кодированием. Например, “cmdlet” 1940 преобразования, который преобразует объекты конвейерной обработки в объекты активной директории (ADO), может декларироваться как “convert/ADO” в командной строке. Аналогичным образом “cmdlet” 1940, который преобразует объекты конвейерной обработки в отделенные запятой значения (csv), может быть декларирован как “convert/csv” в командной строке. Неоторые из “cmdlet” 1940 преобразования (например, convert/XML и convert/html) могут быть командами блокирования, означая, что все объекты конвейерной обработки принимаются перед исполнением преобразования. В типовом случае “cmdlet” 1920 вывода может определять, следует ли использовать информацию форматирования, обеспечиваемую объектами форматирования. Однако если “cmdlet” 1920 преобразования возникает перед “cmdlet” 1920 вывода, то действительное преобразование данных уже произошло, прежде чем “cmdlet” вывода примет объекты. Поэтому в этой ситуации “cmdlet” вывода не может игнорировать преобразование.

Пятая последовательность 1905 иллюстрирует “cmdlet” 1920 форматирования, “cmdlet” 1930 разметки, “cmdlet” 1940 преобразования и “cmdlet” 1910 вывода в указанном порядке. Таким образом, это иллюстрирует, что “cmdlet” 1930 разметки может возникнуть перед “cmdlet” 1940 преобразования.

Шестая последовательность 1906 иллюстрирует “cmdlet” 1920 форматирования, “cmdlet” конкретного преобразования (например, convert/xml cmdlet 1940'), “cmdlet” конкретной трансформации (например, transform/xslt cmdlet 1950) и “cmdlet” 1910 вывода. convert/xml cmdlet 1940' преобразует объекты конвейерной обработки в XML-документ. transform/xslt cmdlet 1950 трансформирует XML-документ в другой XML-документ с использованием XSL (расширяемый язык стилей) таблицы стилей. Процесс трансформирования обычно упоминается как трансформация расширяемого языка стилей (XSLT), в которой XSL-процессор читает XML-документ и следует инструкциям в XSL-таблице стилей для создания нового XML-документа.

Седьмая последовательность 1907 иллюстрирует “cmdlet” 1920 форматирования, “cmdlet” 1930 разметки, “cmdlet” конкретного преобразования (например, convert/xml cmdlet 1940'), “cmdlet” конкретной трансформации (например, transform/xslt cmdlet 1950) и “cmdlet” 1910 вывода. Таким образом, седьмая последовательность 1907 иллюстрирует наличие “cmdlet” 1930 разметки выше по потоку относительно “cmdlet”

преобразования и “cmdlet” трансформации.

Фиг.20 иллюстрирует пример обработки 2000, выполняемой “cmdlet” форматирования. Процесс форматирования начинается в блоке 2001, после того как “cmdlet” форматирования синтаксически проанализирован и активизирован синтаксическим анализатором и процессором конвейерной обработки способом, описанным выше. Затем обработка продолжается в блоке 2002.

В блоке 2002 объект конвейерной обработки принимается в качестве ввода в “cmdlet” форматирования. Затем обработка продолжается в блоке 2004.

В блоке 2004 инициируется запрос для идентификации типа объекта конвейерной обработки. Этот запрос выполняется администратором расширенного типа, как описано выше в связи с фиг.18. После того как администратор расширенного типа идентифицировал тип объекта, обработка продолжается в блоке 2006.

В блоке 2006 идентифицированный тип просматривается в информации отображения. Примерный формат информации отображения показан на фиг.21 и описан ниже. Обработка продолжается в решающем блоке 2008.

В решающем блоке 2008 принимается решение, определен ли идентифицированный тип в информации отображения. Если в информации отображения отсутствует запись для идентифицированного типа, то обработка завершается. В противном случае обработка продолжается в блоке 2010.

В блоке 2010 информация форматирования, связанная с идентифицированным типом, получается из информации отображения. Обработка продолжается в блоке 2012.

В блоке 2012 информация выдается в конвейер. После того как информация выдана, обработка завершается.

Примерная информация, которая может быть выдана, описана ниже более детально. Информация может содержать информацию форматирования, информацию заголовков, нижних колонтитулов и объект сигнализации конца/начала группы. Информация форматирования может включать в себя форму, метку, нумерацию/маркеры списка, ширину столбца, тип кодирующих символов, свойства шрифта содержимого, длину страницы, имя группы-по-свойству и т.п. Каждый из этих параметров может иметь дополнительные спецификации, связанные с ним. Например, форма может определять, является ли форма таблицей, списком или т.п. Метки могут определять, следует ли использовать заголовки столбцов, метки списков или т.п. Кодирование символов может определять ASCII, UTF-8, Unicode и т.п. Свойства шрифта содержимого могут определять шрифт, который применяется к значениям свойств, которые отображаются. Свойство шрифта, устанавливаемого по умолчанию (например, Courier New, 10 point), может использоваться, если свойства шрифта содержимого не определены.

Информация заголовка/нижнего колонтитула может включать в себя объем заголовка/нижнего колонтитула, свойства шрифта, заглавие, подзаголовок, дату, время, нумерацию страниц, разделитель и т.п. Дополнительные свойства могут быть определены для заголовка или нижнего колонтитула. Например, для нижних колонтитулов группы и документа дополнительные свойства могут включать в себя свойства или столбцы для вычисления суммы/итога, отсчеты объектов, строки меток для итогов и отсчетов и т.п.

Объекты сигнализации конца/начала группы выдаются, когда “cmdlet” форматирования обнаруживает, что группа-по-свойству изменилась. Когда это происходит, “cmdlet” форматирования обрабатывает поток конвейерных объектов как

ранее отсортированный и не пересортировывает их. Объекты сигнализации конца/начала группы могут быть вставлены в промежутки между конвейерными объектами. Множество групп по свойствам может быть определено для вложенной сортировки. “cmdlet” форматирования может также выдать объект конца форматирования, который включает в себя финальные суммы и итоги.

Как показано на фиг.21, примерная информация 2100 отображения имеет структурированный формат и содержит информацию (например, информацию форматирования, информацию заголовков/нижних колонтитулов, свойства или методы по группам), связанную с каждым объектом, который был определен. Например, информация 2100 отображения может быть основанной на XML. Каждое из вышеупомянутых свойств может затем определяться в информации отображения. Информация в составе информации 2100 отображения может заполняться владельцем типа объекта, который вводится. Операционная среда обеспечивает определенные API и “cmdlets”, которые позволяют владельцу обновлять информацию отображения путем создания, удаления и модифицирования записей.

На фиг.22 представлена таблица, перечисляющая примерный синтаксис 2201-2213 для некоторых “cmdlets” форматирования (например, format/table, format/list, format/wide), “cmdlets” разметки (например, add/markup), “cmdlets” преобразования (например, convert/text, convert/sv, convert/csv, convert/ADO, convert/XML, convert/html), “cmdlets” трансформации (например, transform/XSLT), “cmdlets” вывода (например, out/console, out/file). Фиг.23 иллюстрирует результаты, визуализированные out/console cmdlet с использованием различных конвейерных последовательностей “cmdlets” обработки вывода (например, “cmdlets” форматирования, “cmdlets” преобразования и “cmdlets” разметки).

Как описано, механизм для получения и применения ограничений в интерактивной среде может быть использован в среде административных инструментальных средств. Однако специалистам в данной области техники должно быть понятно, что механизм может быть использован в различных интерактивных средах.

Хотя выше описаны детали конкретных реализаций и вариантов осуществления, такие детали предназначаются только для удовлетворения требований достаточного раскрытия изобретения, но не для ограничения объема изложенных далее пунктов формулы изобретения. Таким образом, изобретение, как оно определено в формуле изобретения, не ограничено конкретными признаками, описанными выше. Вместо этого изобретение заявлено в любой из его форм или модификаций, которые входят в надлежащий объем прилагаемой формулы изобретения, соответствующим образом интерпретируемый согласно доктрине эквивалентов.

Формула изобретения

1. Машиночитаемый носитель данных, содержащий исполняемые компьютером инструкции, причем инструкции включают в себя

прием строки в интерактивной среде,

идентификацию присваивания атрибутов в строке,

идентификацию логической структуры, связанной с присваиванием атрибутов, и

сохранение информации, которая коррелирует присваивание атрибутов с логической структурой.

2. Машиночитаемый носитель п.1, в котором присваивание атрибутов определяет ограничение для логической структуры.

3. Машиночитаемый носитель п.1, в котором логическая структура включает в себя

переменную, структуру, функцию или сценарий.

4. Машиночитаемый носитель п.1, в котором информация содержит метаданные.

5. Машиночитаемый носитель п.1, дополнительно содержащий применение присвоения атрибутов логической структуре, если логическая структура встречается интерактивным образом.

6. Машиночитаемый носитель п.1, в котором строка включает в себя командную последовательность, вводимую в среду командной строки.

7. Машиночитаемый носитель п.1, в котором строка включает в себя часть сценария.

8. Машиночитаемый носитель п.1, в котором идентификация присвоения атрибутов включает в себя идентификацию множества атрибутов, связанных с логической структурой.

9. Машиночитаемый носитель п.1, в котором присвоение атрибутов определяет тип логической структуры.

10. Машиночитаемый носитель п.1, в котором присвоение атрибутов определяет применение интеллектуального восприятия к логической структуре для автоматического завершения логической структуры.

11. Машиночитаемый носитель п.1, в котором присвоение атрибутов определяет применение указателя предиката к строке, который обеспечивает определение того, продолжается ли обработка строки.

12. Машиночитаемый носитель п.1, в котором присвоение атрибутов определяет применение указателя синтаксического анализа, который предназначен для указания способа для получения логической структуры.

13. Машиночитаемый носитель п.1, в котором присвоение атрибутов определяет указатель генерации данных, который предназначен для генерации набора информации, которая сохранена в логической структуре.

14. Машиночитаемый носитель п.1, в котором присвоение атрибутов определяет указатель проверки данных, который обеспечивает определение того, удовлетворяет ли значение, присвоенное логической структуре, критерию, определенному присваиванием атрибутов.

15. Способ определения ограничений с использованием интерактивной среды для последующего учета этих ограничений в процессе обработки данных, причем способ включает

идентификации предварительно определенного символа начала и символа конца в строке, вводимой в интерактивной среде,

идентификацию имен ограничения между символом начала и символом конца и

идентификацию логической структуры, следующей за символом конца.

16. Способ по п.15, дополнительно содержащий применение ограничения к логической структуре всякий раз, когда логическая структура встречается в интерактивной среде.

17. Способ по п.16, в котором ограничение включает в себя указатель предиката, а применение ограничения включает в себя определение того, удовлетворено ли некоторое условие перед продолжением обработки логической структуры.

18. Способ по п.16, в котором присвоение атрибутов определяет применение интеллектуального восприятия к логической структуре для автоматического завершения логической структуры.

19. Способ по п.16, в котором присвоение атрибутов определяет применение указателя синтаксического анализа, который действует для указания способа

получения логической структуры.

20. Способ по п.16, в котором присвоение атрибутов определяет указатель генерации данных, который действует для генерации набора информации, которая сохраняется в логической структуре.

21. Способ по п.16, в котором присвоение атрибутов определяет указатель подтверждения данных, который действует для определения того, удовлетворяет ли значение, присвоенное логической структуре, критерию, определенному присвоением атрибутов.

22. Способ по п.15, в котором символ начала содержит открывающую скобку, а символ конца содержит закрывающую скобку.

23. Система для обработки параметров ввода, содержащая

средство для обработки,

средство памяти, причем средство памяти выделено для множества исполняемых компьютером инструкций, которые загружаются в средство памяти для исполнения средством для обработки, причем исполняемые компьютером инструкции выполняют способ, включающий в себя

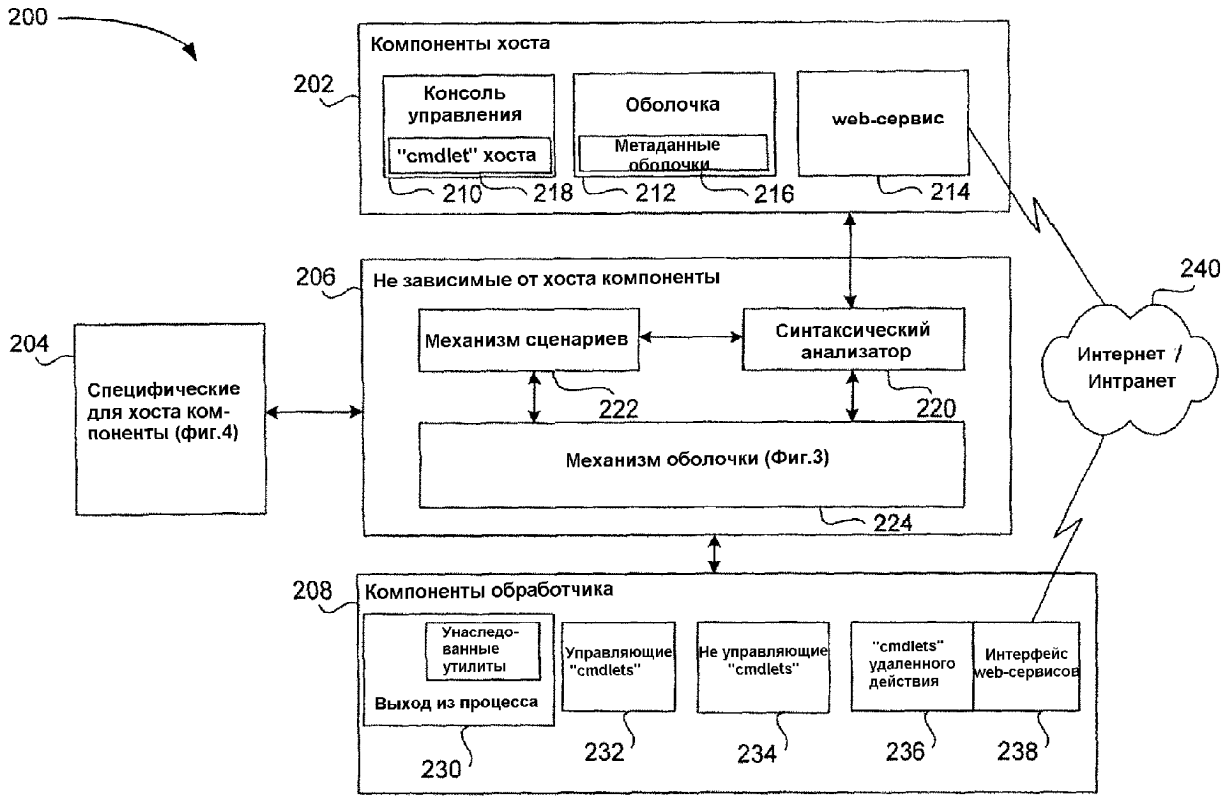
средство для приема строки в интерактивной среде,

средство для идентификации присваивания атрибутов в строке,

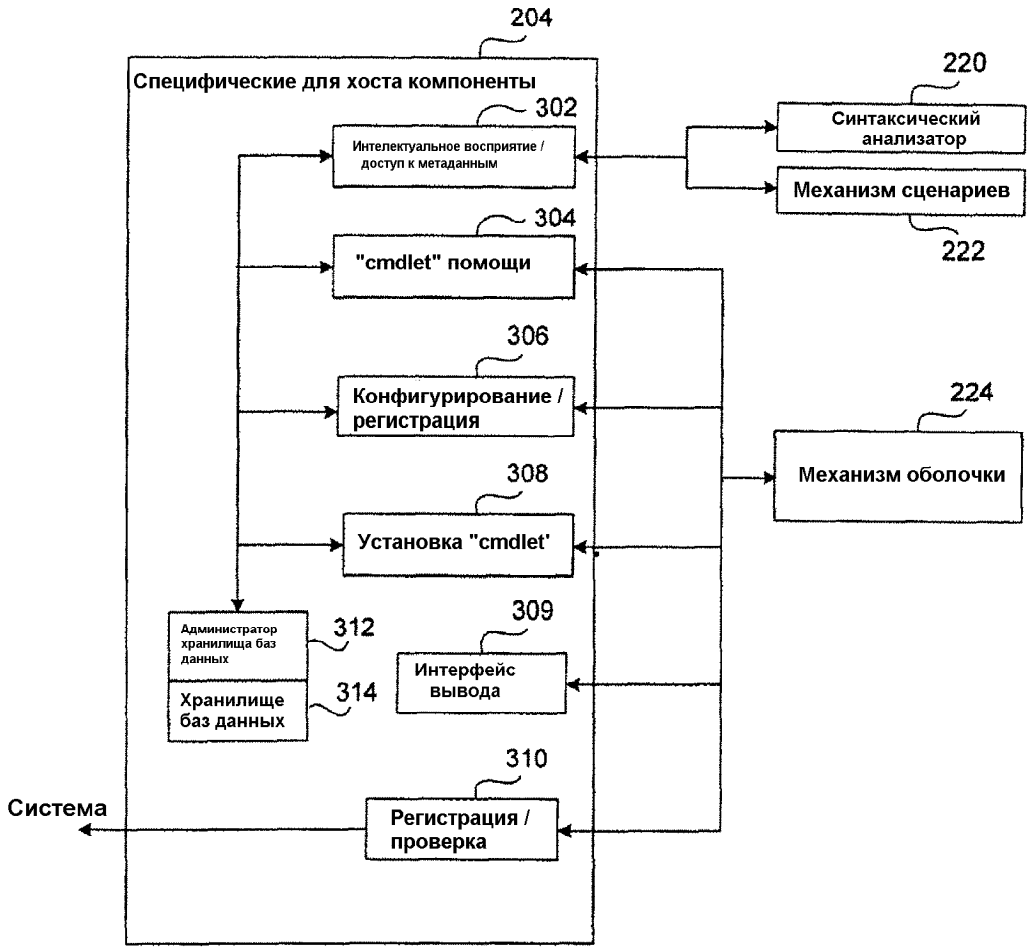
средство для идентификации логической структуры, связанной с присваиванием атрибутов, и

средство для сохранения информации, которая коррелирует присваивание атрибутов с логической структурой.

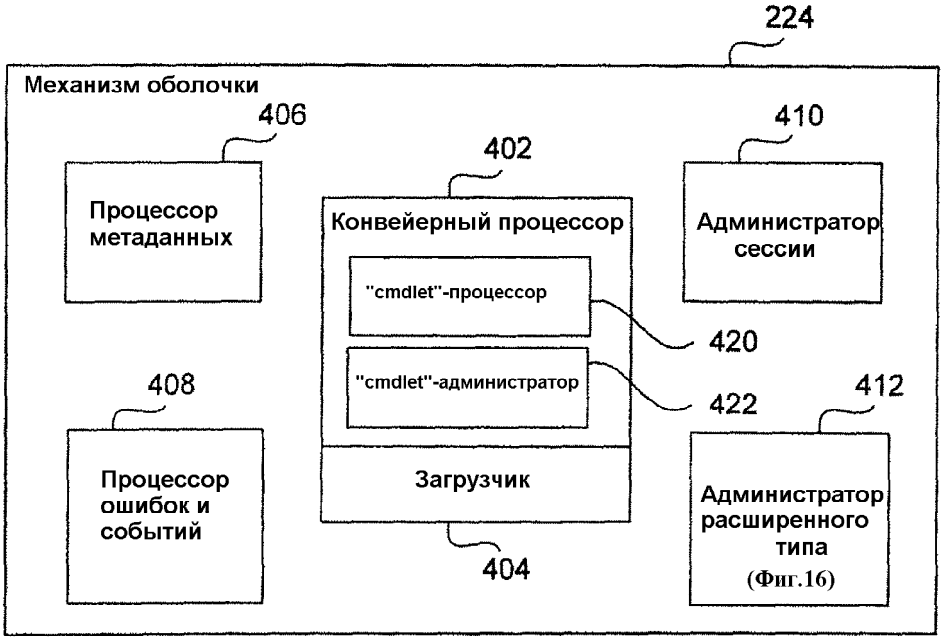
24. Система по п.23, дополнительно содержащая средство для применения присваивания атрибутов к логической структуре, когда логическая структура встречается интерактивным образом.


200
Компоненты хоста
202
Консоль управления
"cmdlet" хоста
Оболочка
Метаданные оболочки
web-сервис
210
218
212
216
214
206
Не зависимые от хоста компоненты
Механизм сценариев
Синтаксический анализатор
222
220
Механизм оболочки (Фиг.3)
224
204
Специфические для хоста компоненты (фиг.4)
240
Интернет / Интранет
208
Компоненты обработчика
Унаследованные утилиты
Выход из процесса
Управляющие "cmdlets"
Не управляющие "cmdlets"
"cmdlets" удаленного действия
Интерфейс web-сервисов
230
232
234
236
238

Фиг.2

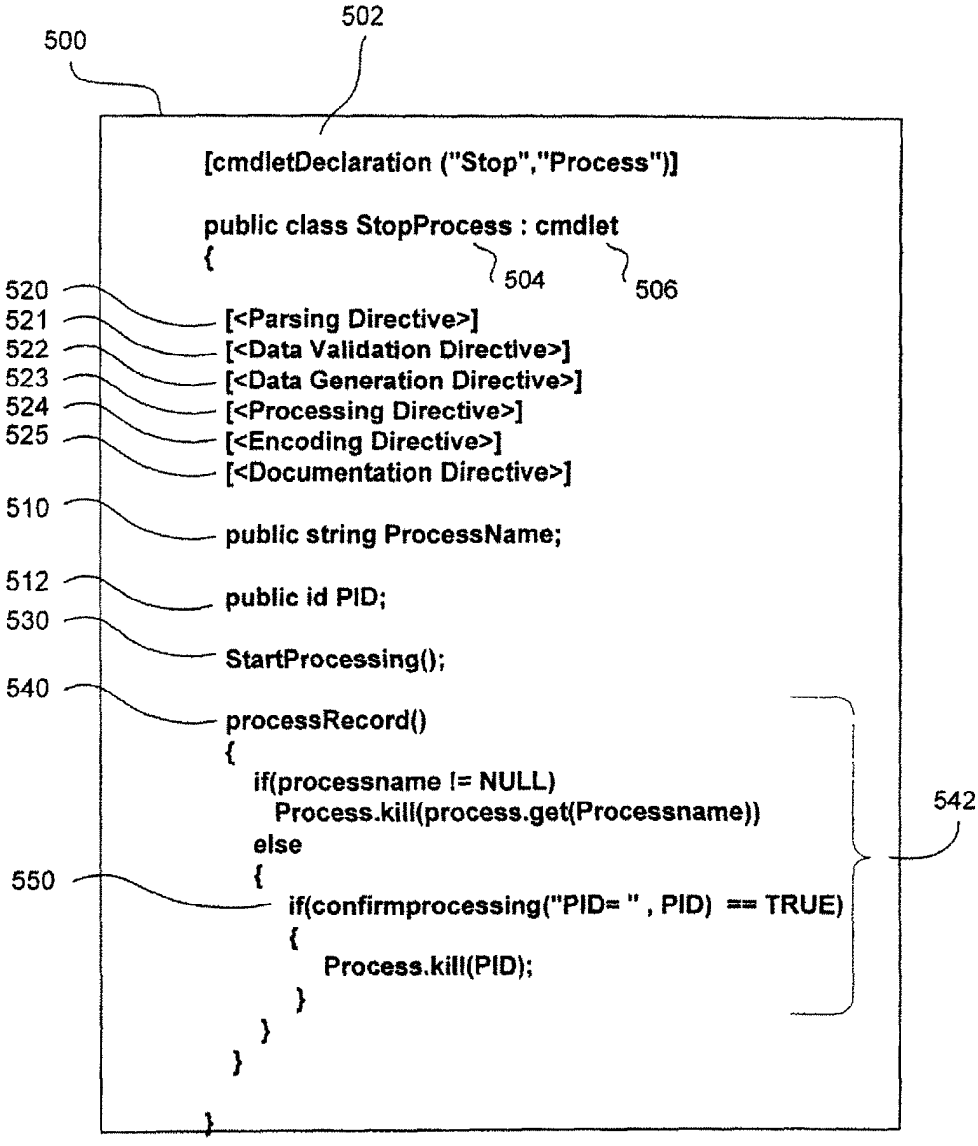

204
Специфические для хоста компоненты
302
Интелектуальное восприятие / доступ к метаданным
304
"cmdlet" помощи
306
Конфигурирование / регистрация
308
Установка "cmdlet'
Администратор хранилища баз данных
312
Хранилище баз данных
314
309
Интерфейс вывода
310
Регистрация / проверка
Система
220
Синтаксический анализатор
Механизм сценариев
222
224
Механизм оболочки

Фиг.3

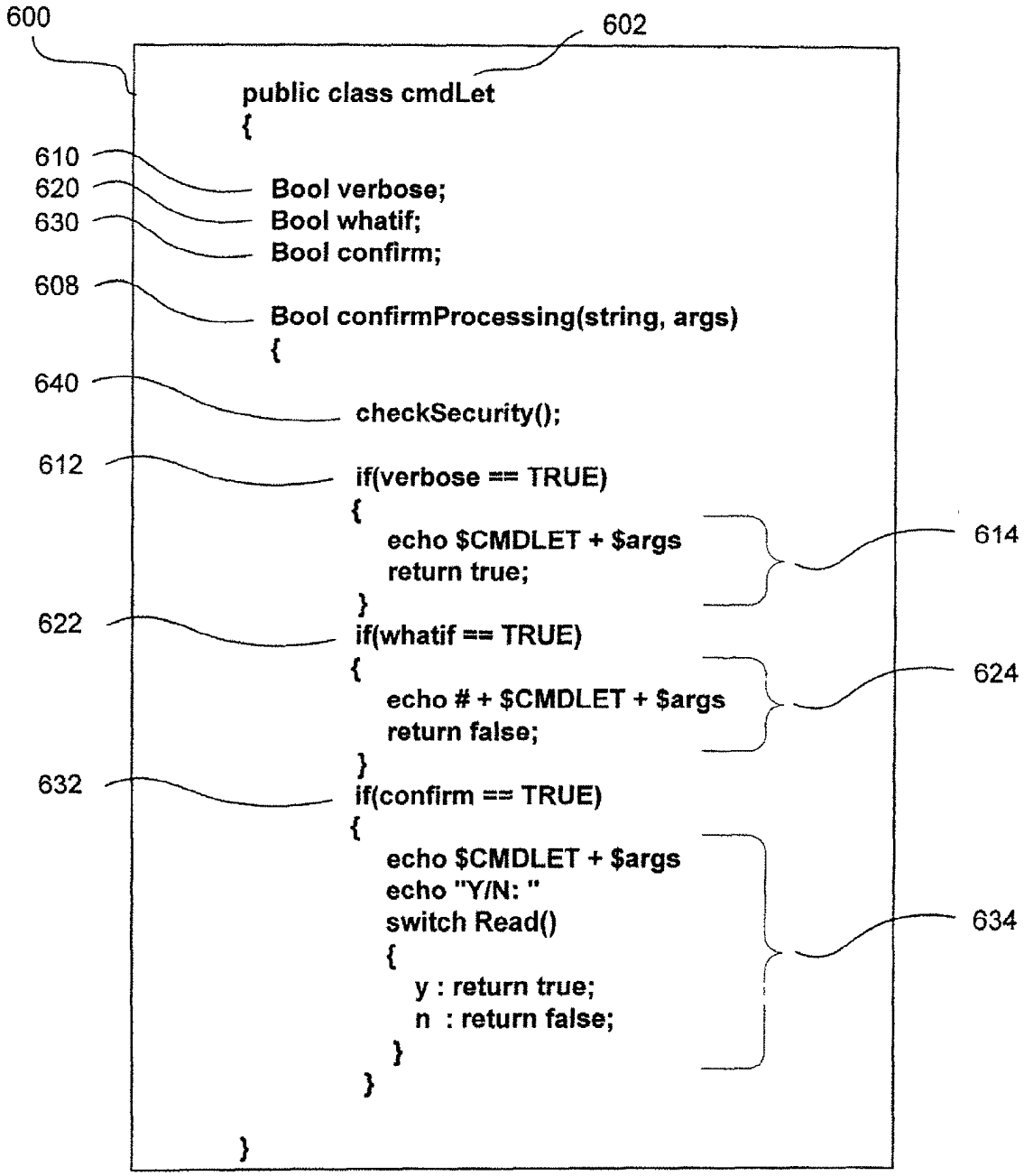


(Фиг.16)

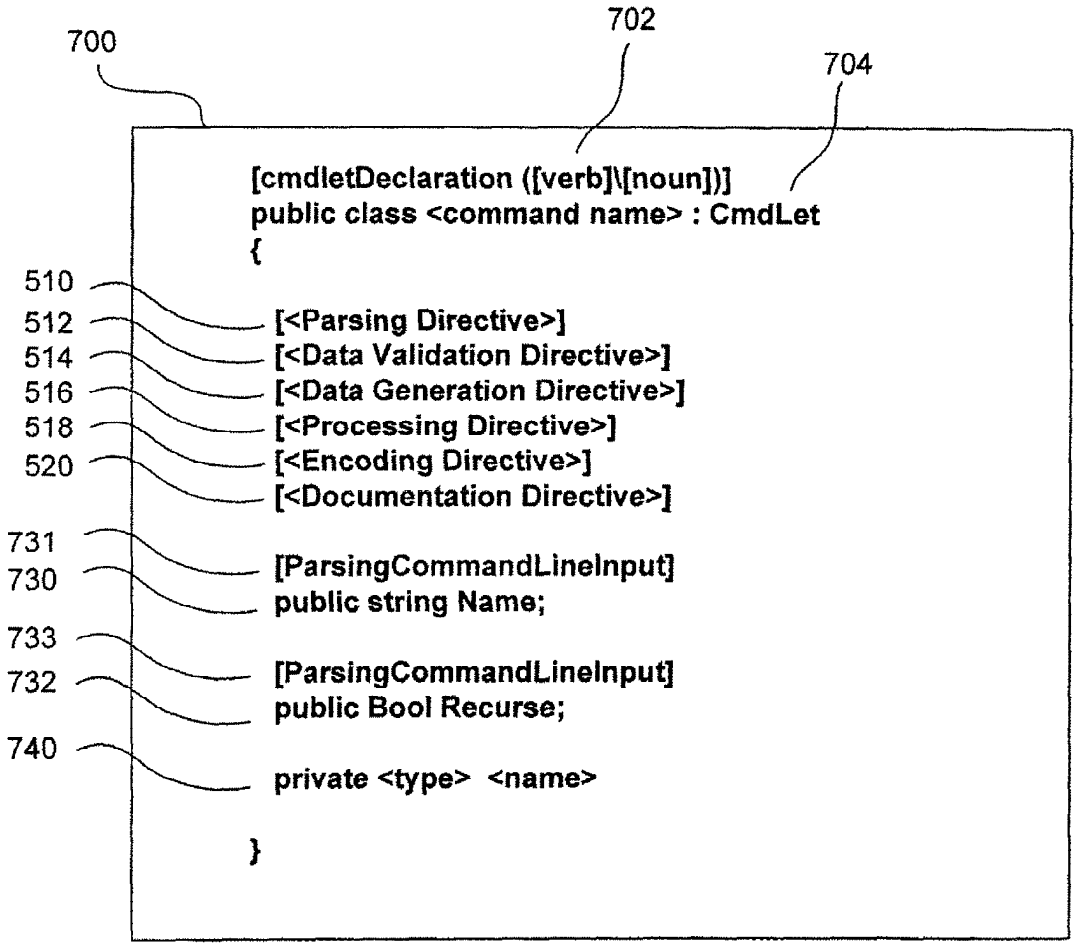
Фиг.4



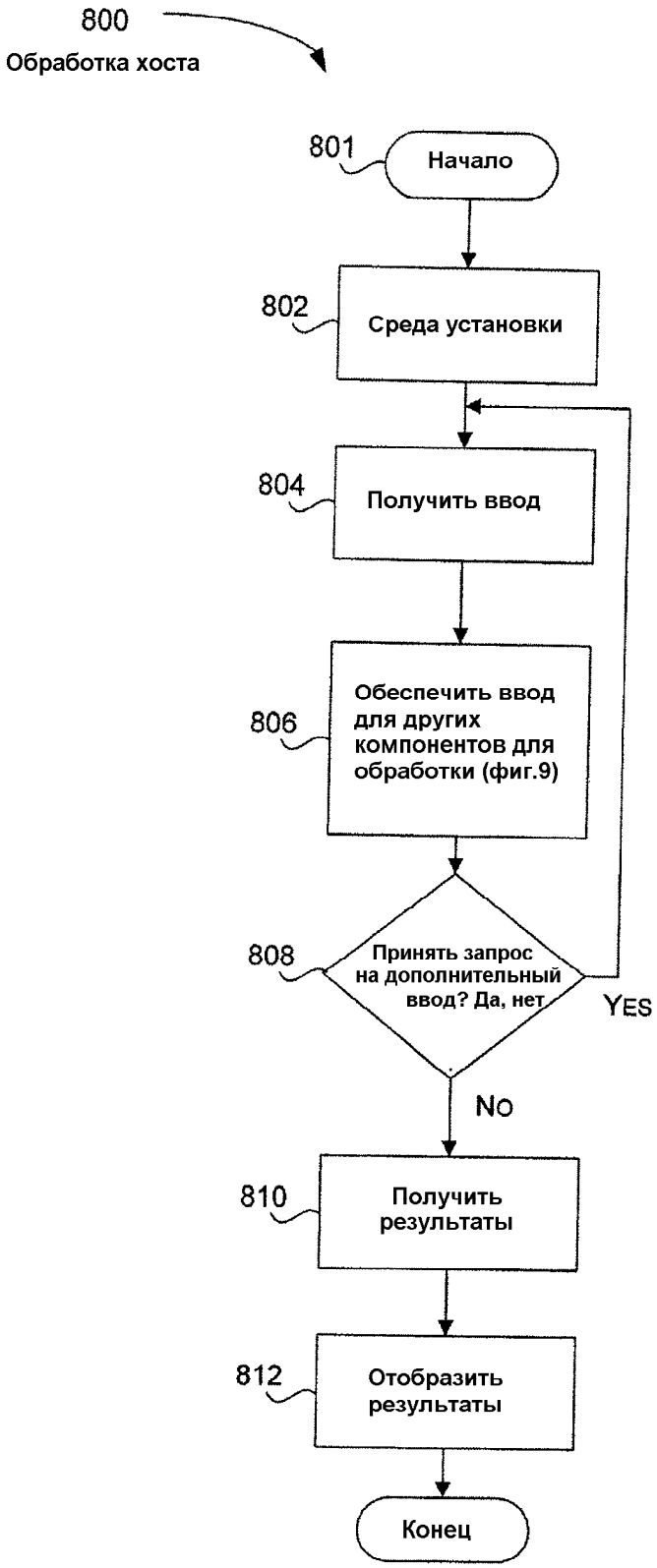
Фиг.5


600
602
public class cmdLet
{
610
Bool verbose;
620
Bool whatif;
630
Bool confirm;
608
Bool confirmProcessing(string, args)
{
640
checkSecurity();
612
if(verbose == TRUE)
{
echo $CMDLET + $args
return true;
}
614
622
if(whatif == TRUE)
{
echo # + $CMDLET + $args
return false;
}
624
632
if(confirm == TRUE)
{
echo $CMDLET + $args
echo "Y/N: "
switch Read()
{
y : return true;
n : return false;
}
}
634
}

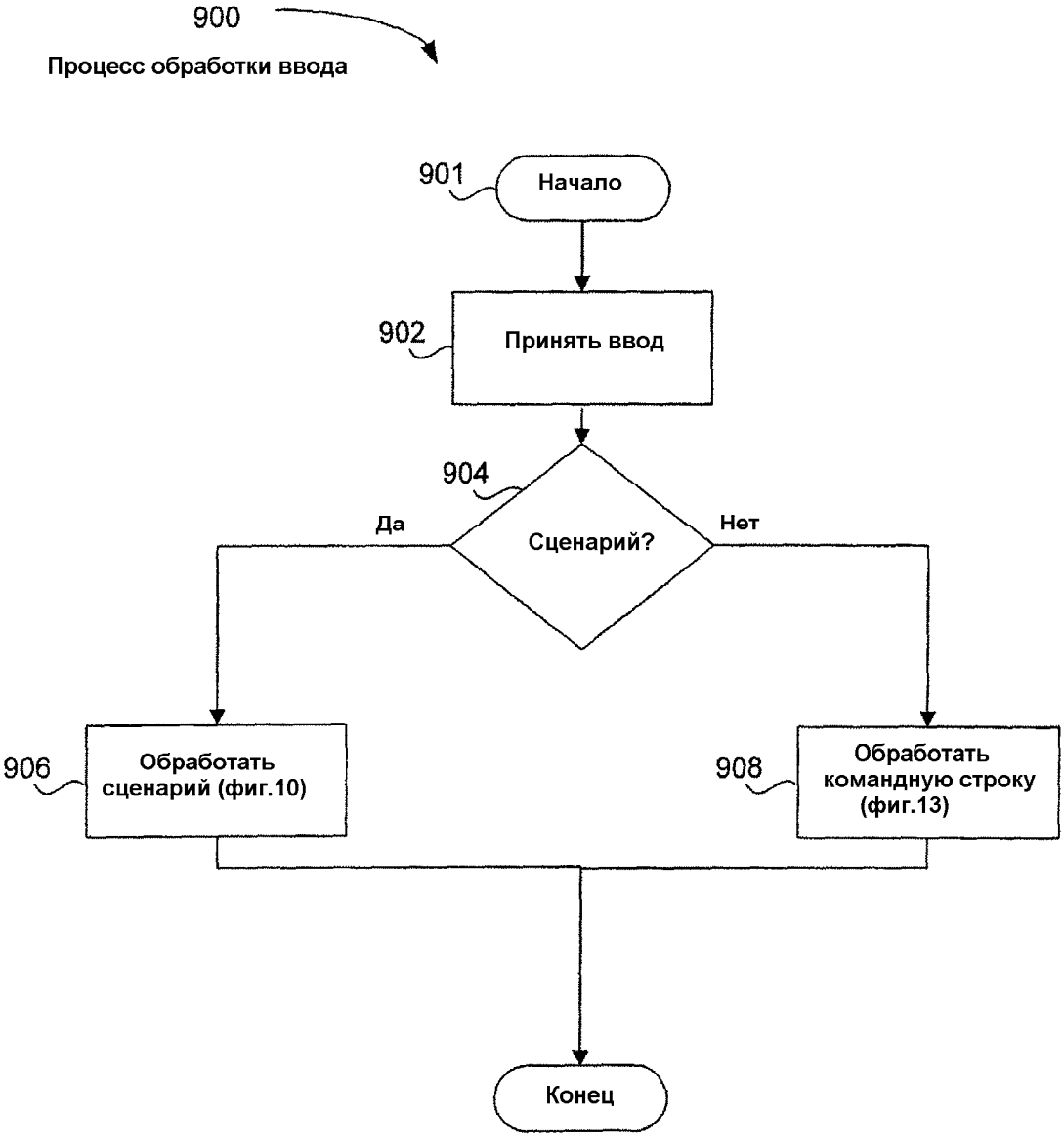
Фиг.6


700
702
704
[cmdletDeclaration ([verb]\[noun])]
public class <command name> : CmdLet
{
510
512
514
516
518
520
[<Parsing Directive>]
[<Data Validation Directive>]
[<Data Generation Directive>]
[<Processing Directive>]
[<Encoding Directive>]
[<Documentation Directive>]
731
730
733
732
740
[ParsingCommandLineInput]
public string Name;
[ParsingCommandLineInput]
public Bool Recurse;
private <type> <name>
}

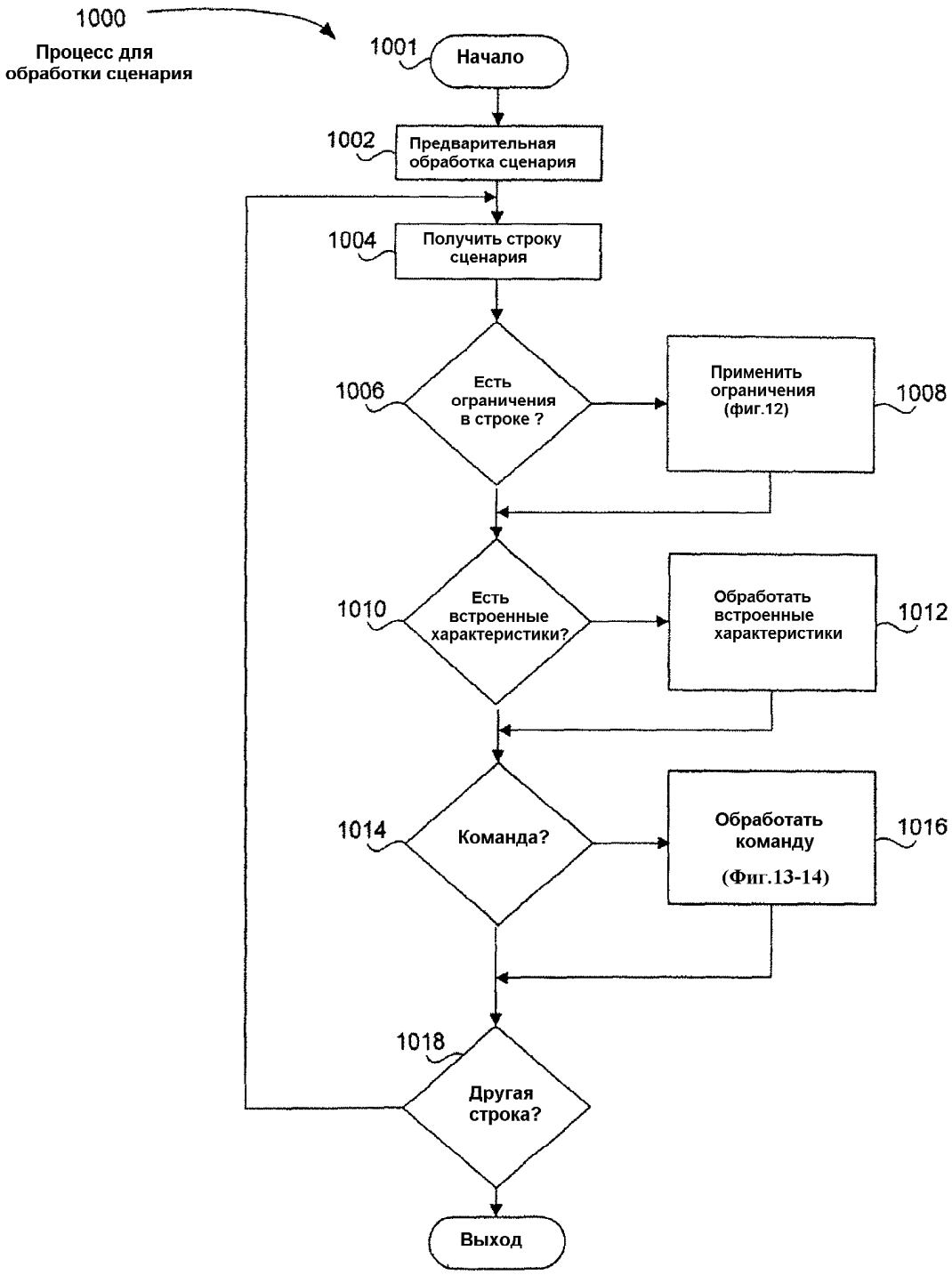
Фиг.7


800
Обработка хоста
801
Начало
802
Среда установки
804
Получить ввод
806
Обеспечить ввод для других компонентов для обработки (фиг.9)
808
Принять запрос на дополнительный ввод? Да, нет
YES
NO
810
Получить результаты
812
Отобразить результаты
Конец

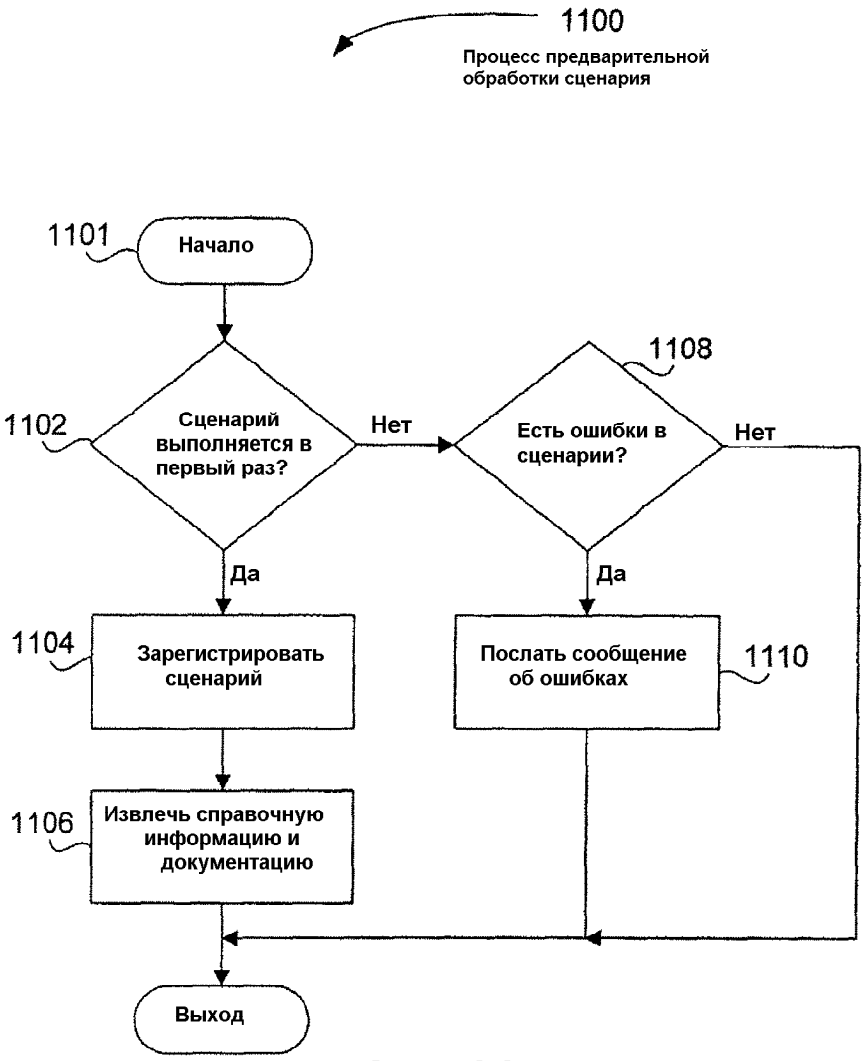
Фиг.8


900
Процесс обработки ввода
901
Начало
902
Принять ввод
904
Сценарий?
Да
Нет
906
Обработать сценарий (фиг.10)
908
Обработать командную строку (фиг.13)
Конец

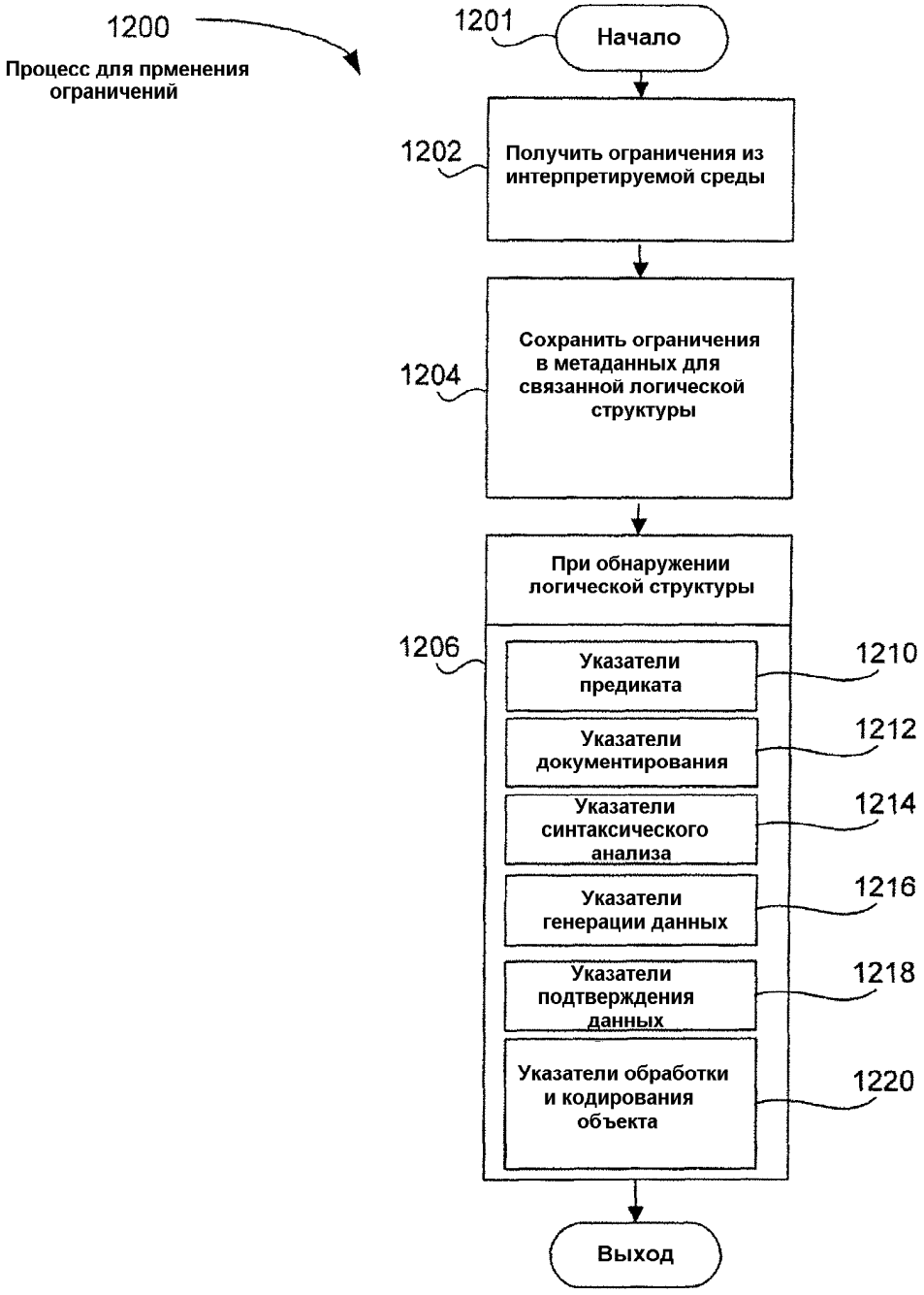
Фиг.9


1000
Процесс для обработки сценария
1001
Начало
1002
Предварительная обработка сценария
1004
Получить строку сценария
1006
Есть ограничения в строке ?
Применить ограничения (фиг.12)
1008
1010
Есть встроенные характеристики?
Обработать встроенные характеристики
1012
1014
Команда?
Обработать команду
(Фиг.13-14)
1016
1018
Другая строка?
Выход

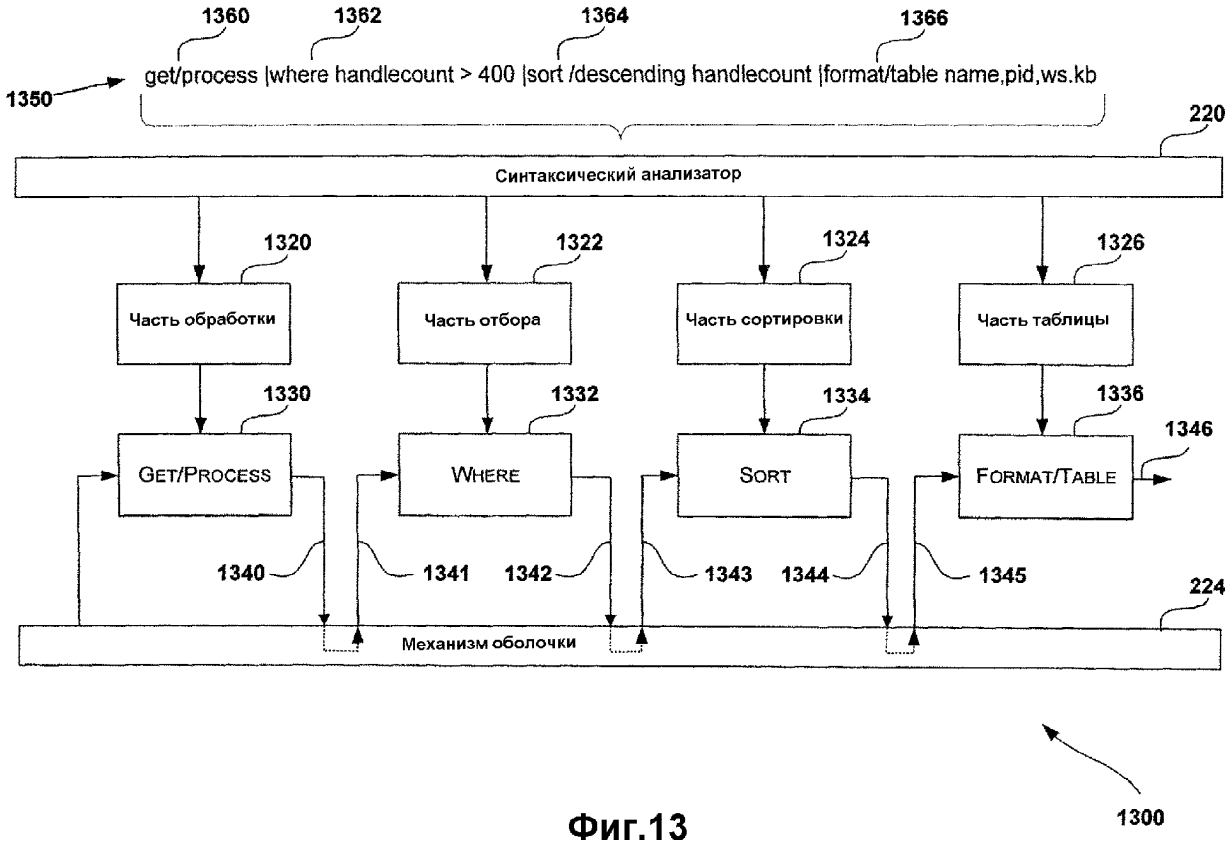
Фиг.10


1100
Процесс предварительной обработки сценария
1101
Начало
1102
Сценарий выполняется в первый раз?
Нет
1108
Есть ошибки в сценарии?
Нет
Да
Да
1104
Зарегистрировать сценарий
Послать сообщение об ошибках
1110
1106
Извлечь справочную информацию и документацию
Выход

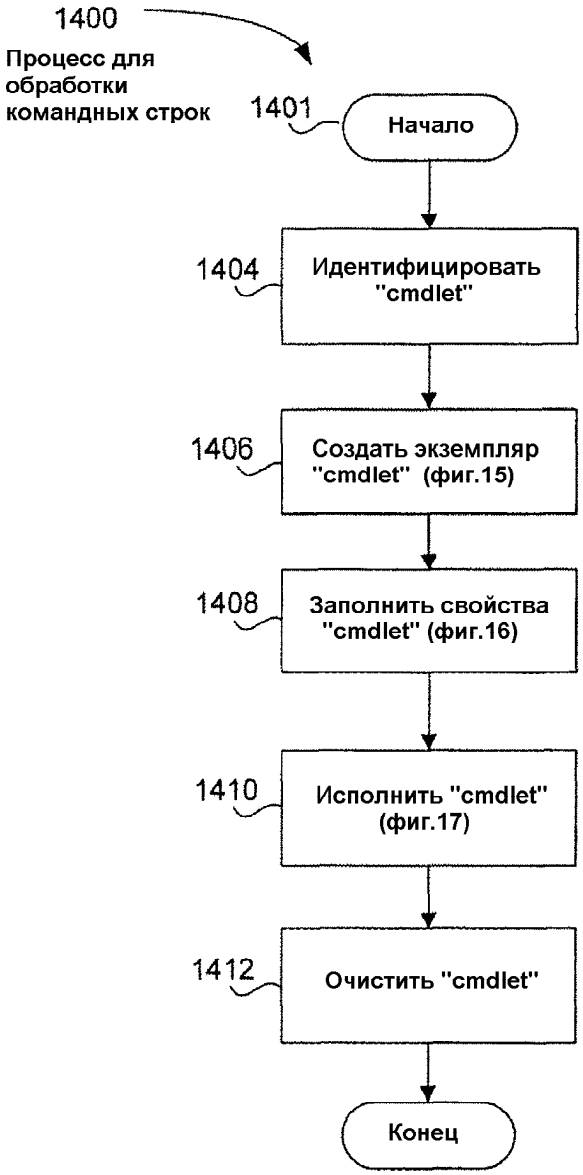
Фиг.11


1200
Процесс для пременения ограничений
1201
Начало
1202
Получить ограничения из интерпретируемой среды
1204
Сохранить ограничения в метаданных для связанной логической структуры
При обнаружении логической структуры
1206
Указатели предиката
1210
Указатели документирования
1212
Указатели синтаксического анализа
1214
Указатели генерации данных
1216
Указатели подтверждения данных
1218
Указатели обработки и кодирования объекта
1220
Выход

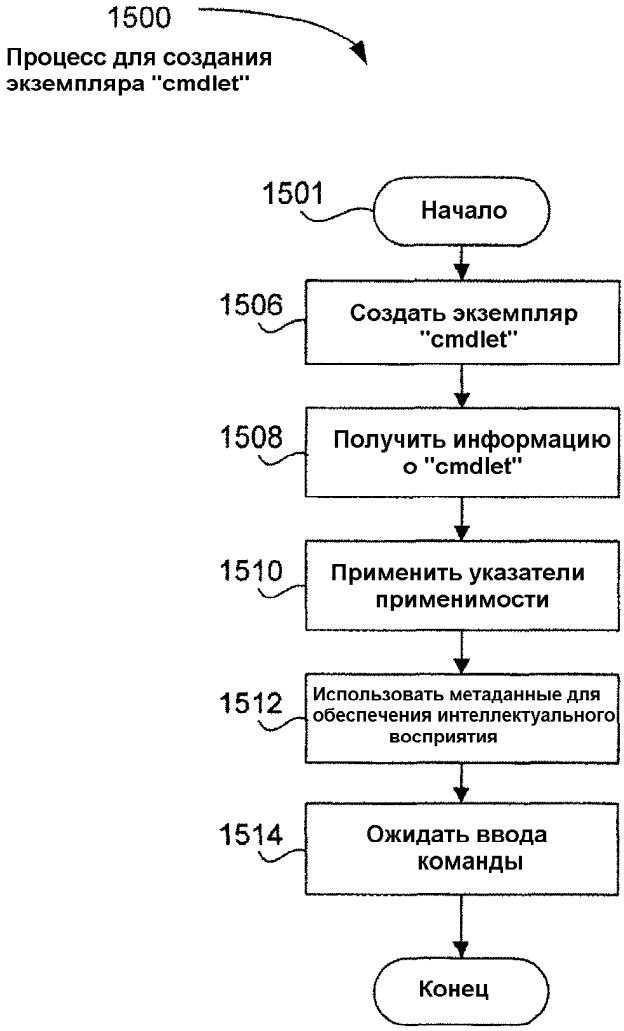
Фиг.12


1360
1362
1364
1366
1350
get/process |where handlecount > 400 |sort /descending handlecount |format/table name,pid,ws.kb
220
Синтаксический анализатор
1320
1322
1324
1326
Часть обработки
Часть отбора
Часть сортировки
Часть таблицы
1330
1332
1334
1336
1346
GET/PROCESS
WHERE
SORT
FORMAT/TABLE
1340
1341
1342
1343
1344
1345
224
Механизм оболочки
1300

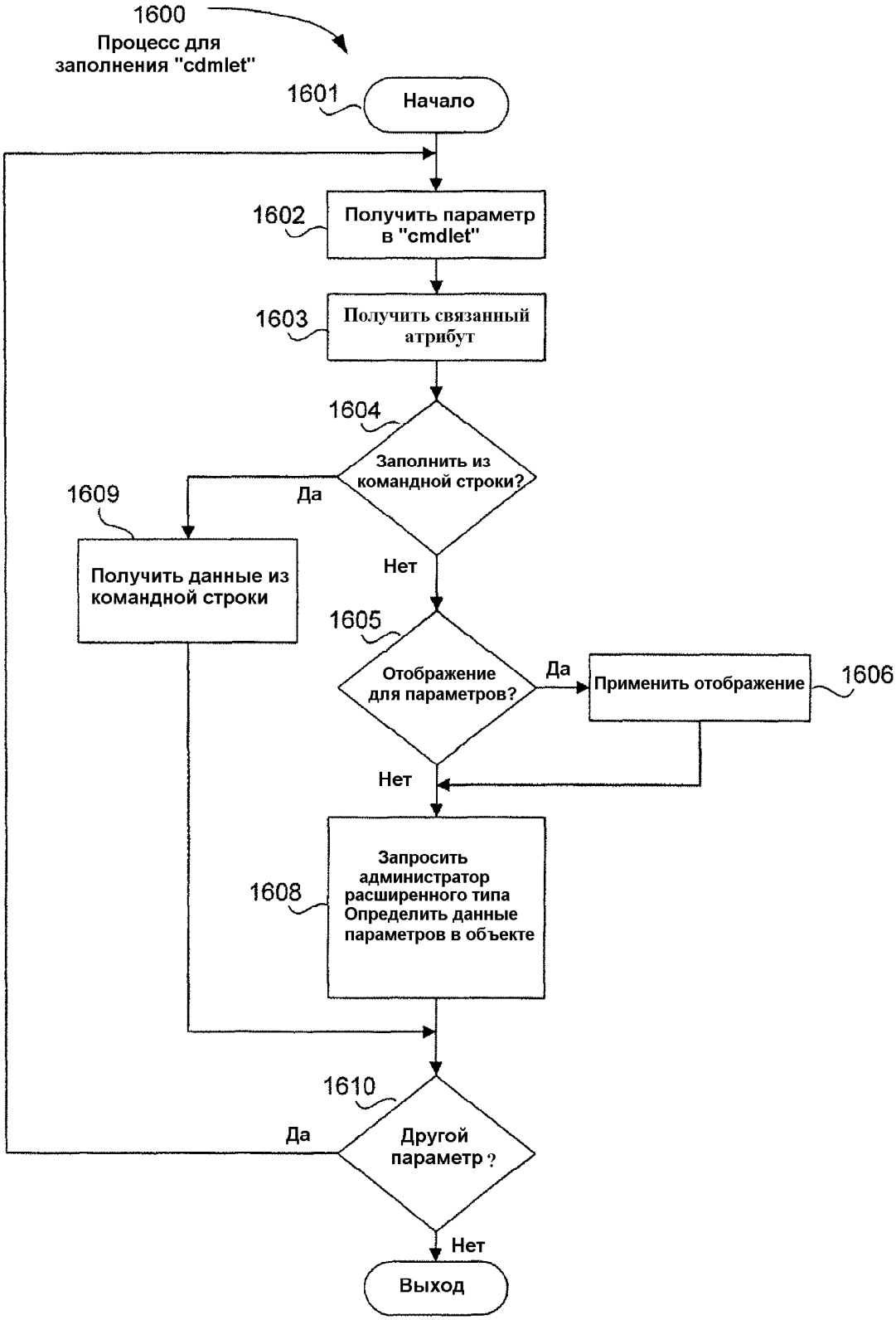
Фиг.13


1400
Процесс для обработки командных строк
1401
Начало
1404
Идентифицировать "cmdlet"
1406
Создать экземпляр "cmdlet" (фиг.15)
1408
Заполнить свойства "cmdlet" (фиг.16)
1410
Исполнить "cmdlet" (фиг.17)
1412
Очистить "cmdlet"
Конец

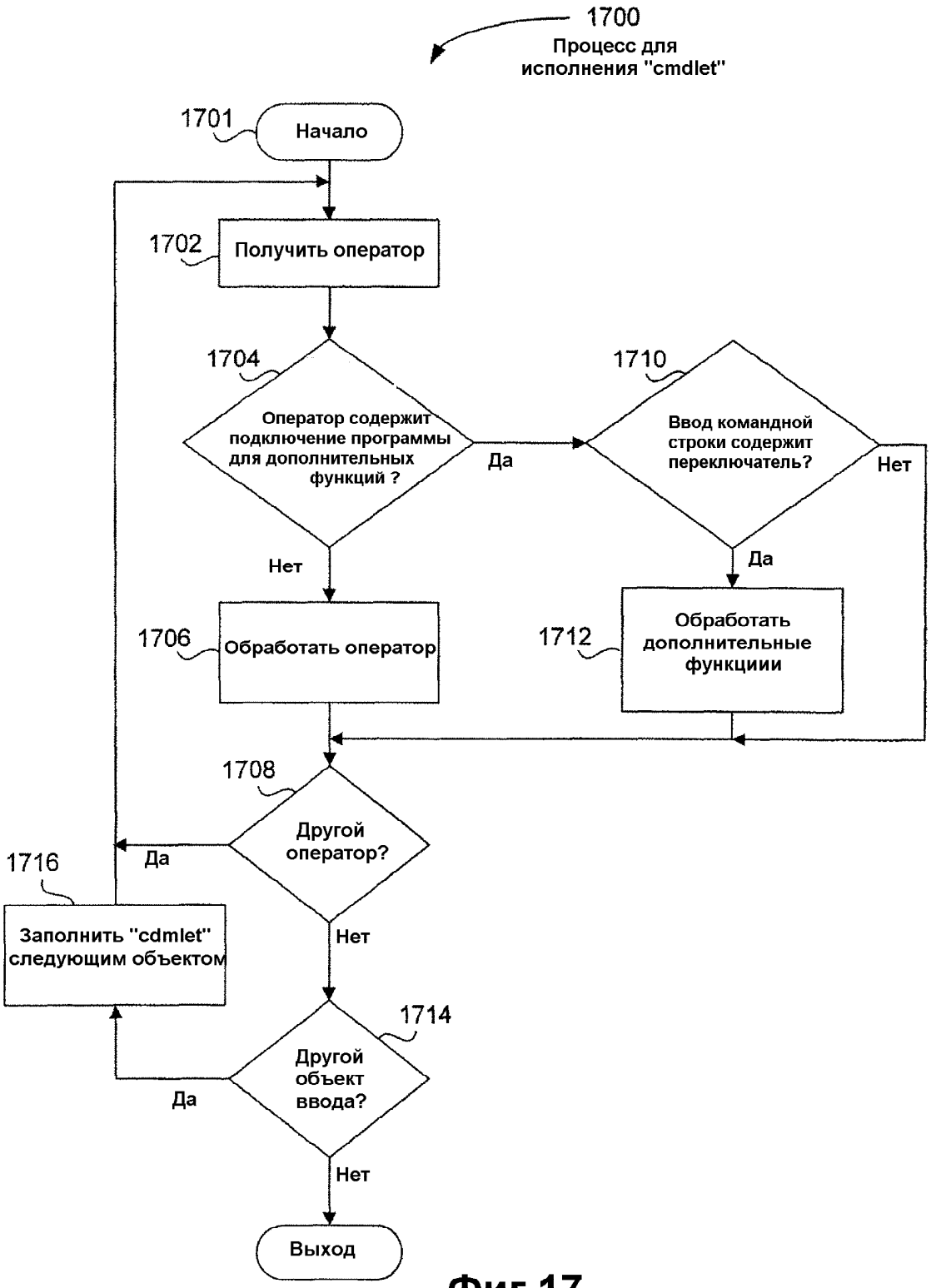
Фиг.14


1500
Процесс для создания экземпляра "cmdlet"
1501
Начало
1506
Создать экземпляр "cmdlet"
1508
Получить информацию о "cmdlet"
1510
Применить указатели применимости
1512
Использовать метаданные для обеспечения интеллектуального восприятия
1514
Ожидать ввода команды
Конец

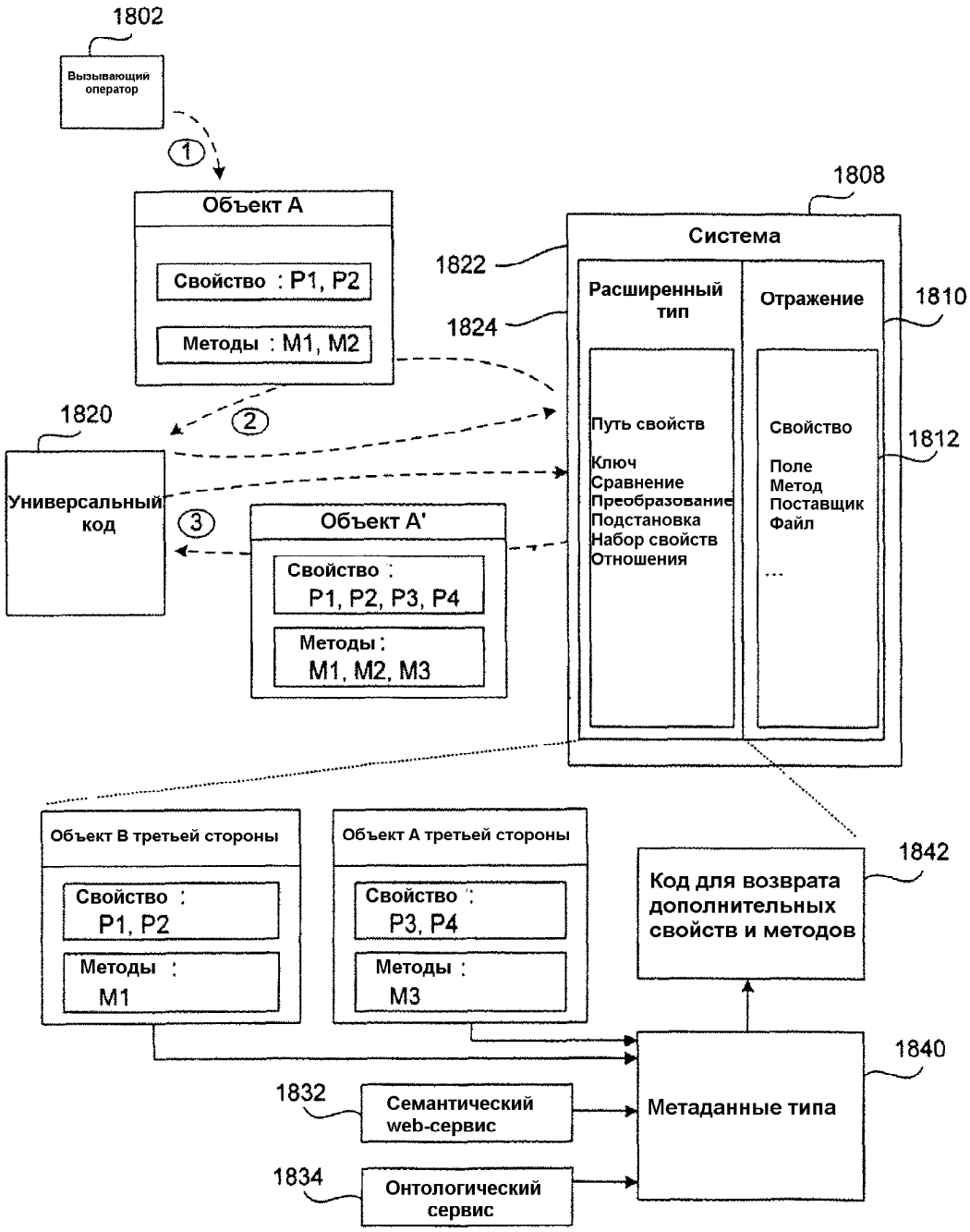
Фиг.15


1600
Процесс для заполнения "cdmlet"
1601
Начало
1602
Получить параметр в "cmdlet"
1603
Получить связанный атрибут
1604
Заполнить из командной строки?
Да
1609
Получить данные из командной строки
Нет
1605
Отображение для параметров?
Да
Применить отображение
1606
Нет
1608
Запросить администратор расширенного типа Определить данные параметров в объекте
1610
Другой параметр?
Да
Нет
Выход

Фиг.16

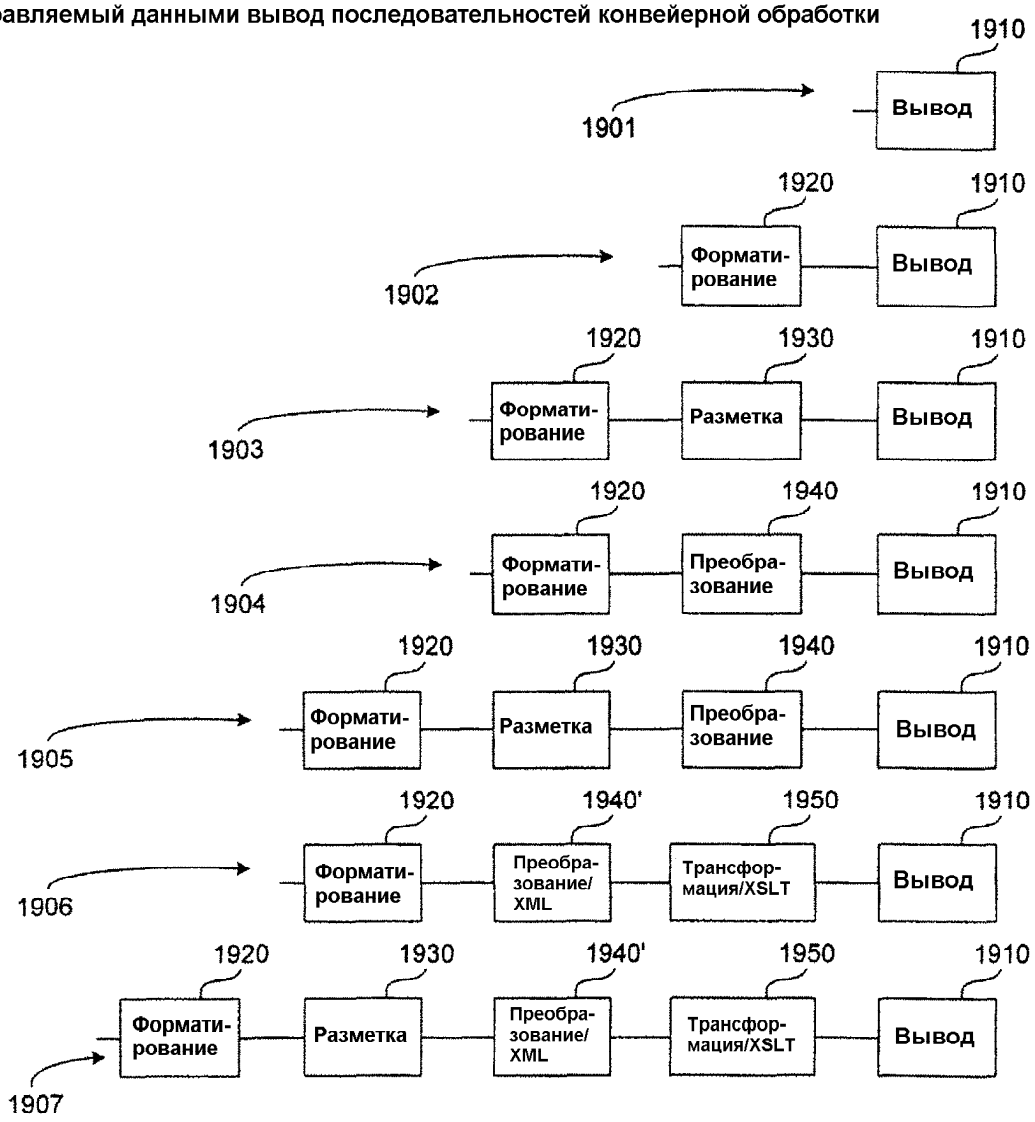

1700
Процесс для исполнения "cmdlet"
1701
Начало
1702
Получить оператор
1704
Оператор содержит подключение программы для дополнительных функций ?
Да
Нет
1710
Ввод командной строки содержит переключатель?
Да
Нет
1706
Обработать оператор
1712
Обработать дополнительные функциии
1708
Другой оператор?
Да
Нет
1716
Заполнить "cdmlet" следующим объектом
1714
Другой объект ввода?
Да
Нет
Выход

Фиг.17

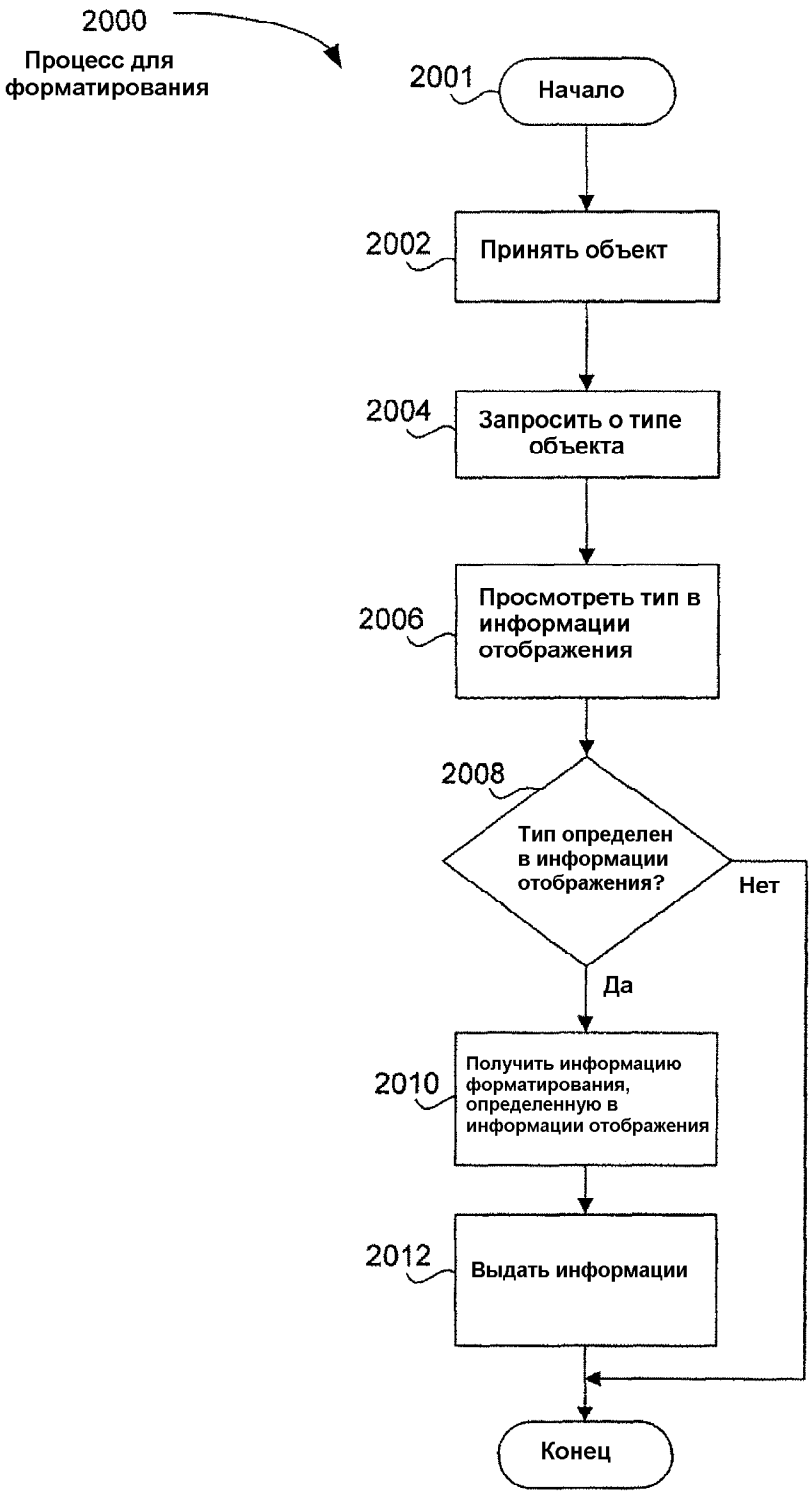

1802
Вызывающий оператор
1
Объект A
Свойство : P1, P2
Методы : M1, M2
1808
Система
1822
1824
Расширенный тип
Отражение
1810
Путь свойств
Ключ
Сравнение
Преобразование
Подстановка
Набор свойств
Отношения
Свойство
Поле
Метод
Поставщик
Файл
...
1812
1820
Универсальный код
2
3
Объект A'
Свойство : P1, P2, P3, P4
Методы : M1, M2, M3
Объект B третьей стороны
Свойство : P1, P2
Методы : M1
Объект A третьей стороны
Свойство : P3, P4
Методы : M3
1842
Код для возврата дополнительных свойств и методов
1840
Метаданные типа
1832
Семантический web-сервис
1834
Онтологический сервис

Фиг.18

Управляемый данными вывод последовательностей конвейерной обработки



Фиг.19

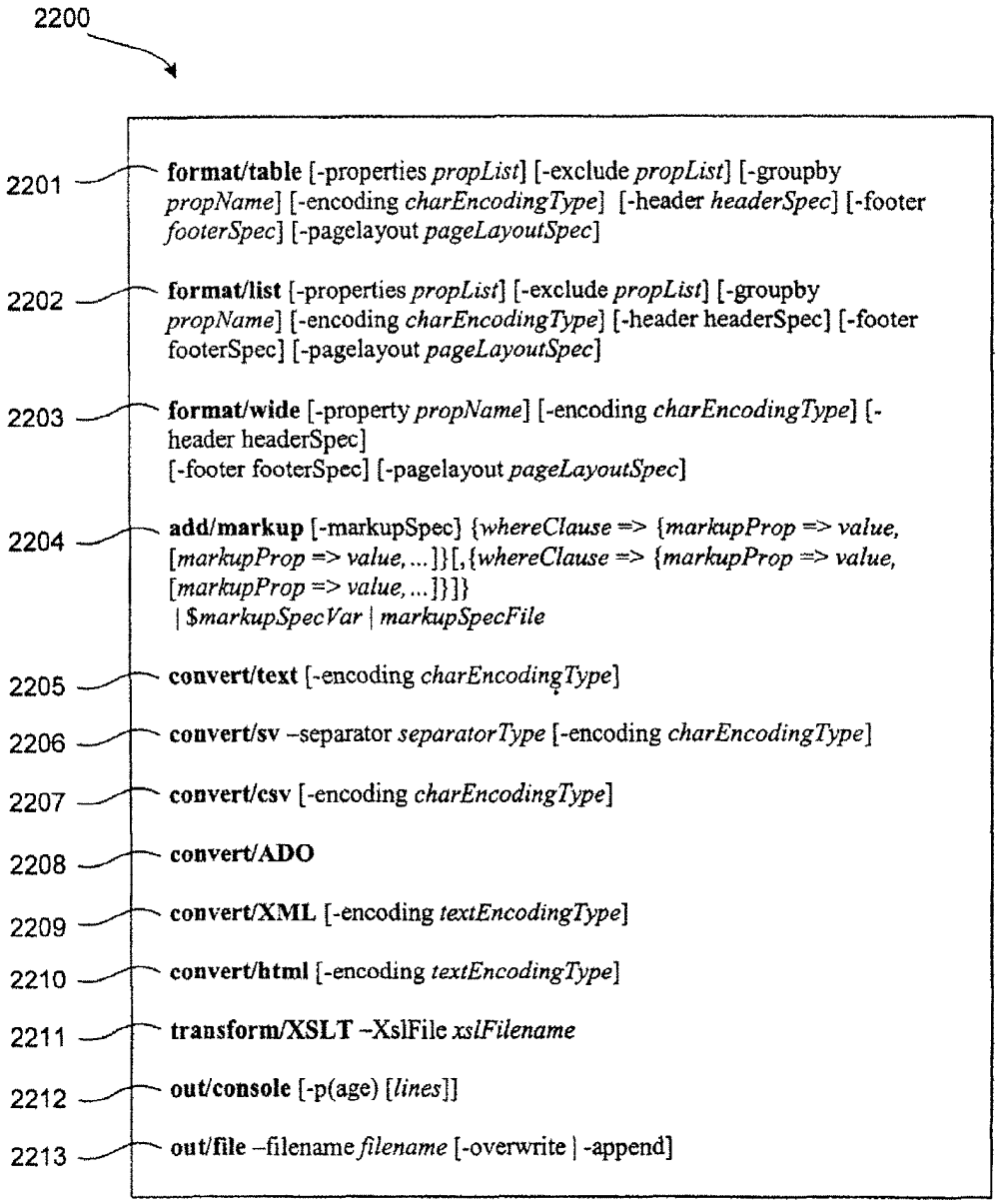

2000
Процесс для форматирования
2001
Начало
2002
Принять объект
2004
Запросить о типе объекта
2006
Просмотреть тип в информации отображения
2008
Тип определен в информации отображения?
Нет
Да
2010
Получить информацию форматирования, определенную в информации отображения
2012
Выдать информации
Конец

Фиг.20

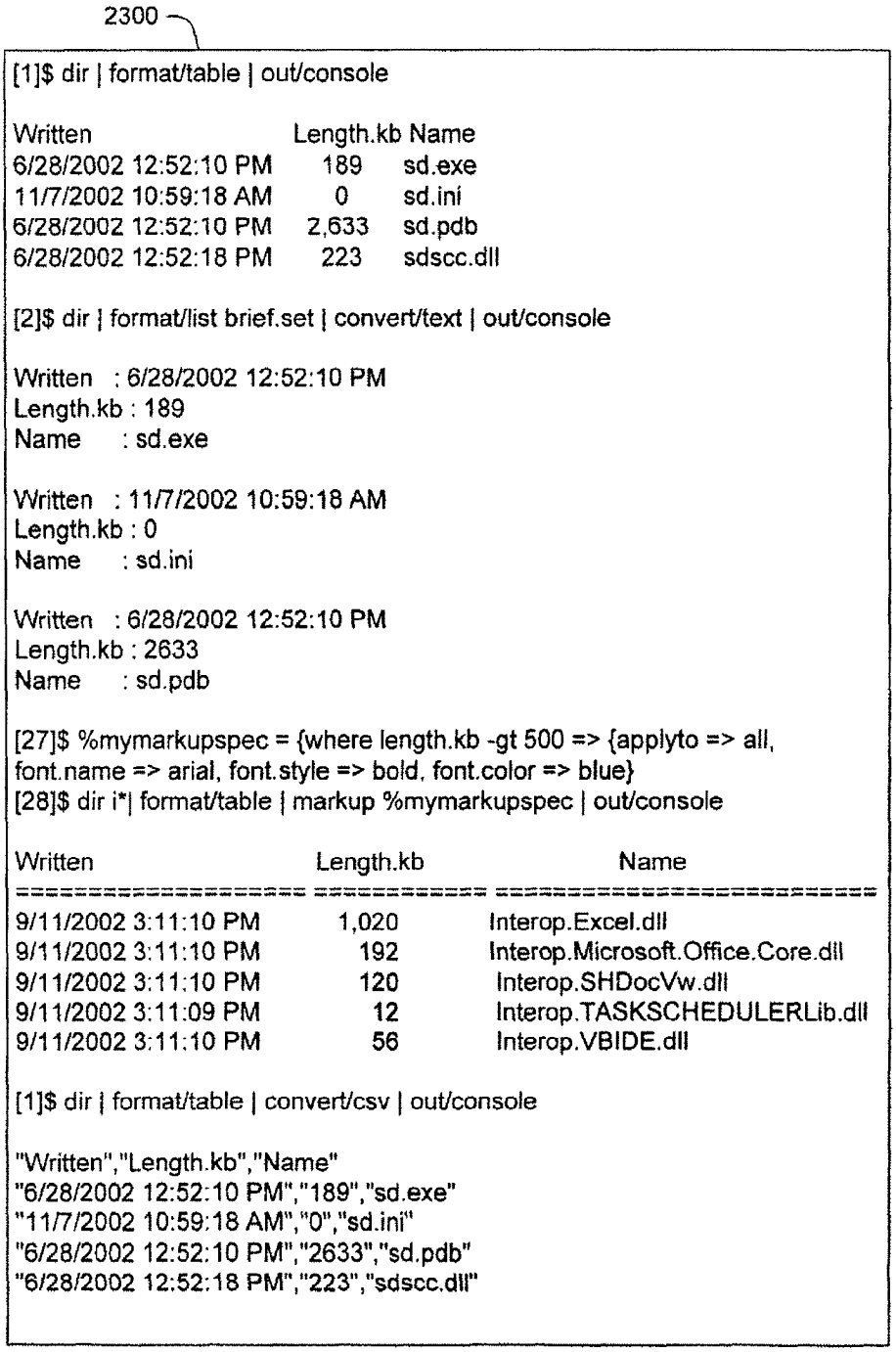
2100
Пример информации отображения

```
<DisplayInfo>
 <Type> DirectoryInfo </Type>
 <Shape> Table </Shape>
 <Columns>
  <Column>
   <PropertyName> LastWriteTime </PropertyName>
  </Column>
  <Column>
   <PropertyName> Size </PropertyName>
   <Align> left </Align>
   <Width> 10 </Width>
   <FormatString> D </FormatString>
   <Label> SIZE </Label>
  </Column>
 </Columns>
 <Header>
  ...
 </Header>
 <GroupBy> ParentPath </GroupBy>
 ...
</DisplayInfo>
```

Фиг.21


2200
2201
format/table [-properties propList] [-exclude propList] [-groupby propName] [-encoding charEncodingType] [-header headerSpec] [-footer footerSpec] [-pagelayout pageLayoutSpec]
2202
format/list [-properties propList] [-exclude propList] [-groupby propName] [-encoding charEncodingType] [-header headerSpec] [-footer footerSpec] [-pagelayout pageLayoutSpec]
2203
format/wide [-property propName] [-encoding charEncodingType] [-header headerSpec] [-footer footerSpec] [-pagelayout pageLayoutSpec]
2204
add/markup [-markupSpec] {whereClause => {markupProp => value, [markupProp => value, ...]}[,{whereClause => {markupProp => value, [markupProp => value, ...]}]} | $markupSpecVar | markupSpecFile
2205
convert/text [-encoding charEncodingType]
2206
convert/sv –separator separatorType [-encoding charEncodingType]
2207
convert/csv [-encoding charEncodingType]
2208
convert/ADO
2209
convert/XML [-encoding textEncodingType]
2210
convert/html [-encoding textEncodingType]
2211
transform/XSLT –XslFile xslFilename
2212
out/console [-p(age) [lines]]
2213
out/file –filename filename [-overwrite | -append]

Фиг.22


2300
[1]$ dir | format/table | out/console
Written Length.kb Name
6/28/2002 12:52:10 PM 189 sd.exe
11/7/2002 10:59:18 AM 0 sd.ini
6/28/2002 12:52:10 PM 2,633 sd.pdb
6/28/2002 12:52:18 PM 223 sdscc.dll
[2]$ dir | format/list brief.set | convert/text | out/console
Written : 6/28/2002 12:52:10 PM
Length.kb : 189
Name : sd.exe
Written : 11/7/2002 10:59:18 AM
Length.kb : 0
Name : sd.ini
Written : 6/28/2002 12:52:10 PM
Length.kb : 2633
Name : sd.pdb
[27]$ %mymarkupspec = {where length.kb -gt 500 => {applyto => all,
font.name => arial, font.style => bold, font.color => blue}
[28]$ dir i*| format/table | markup %mymarkupspec | out/console
Written Length.kb Name
9/11/2002 3:11:10 PM 1,020 Interop.Excel.dll
9/11/2002 3:11:10 PM 192 Interop.Microsoft.Office.Core.dll
9/11/2002 3:11:10 PM 120 Interop.SHDocVw.dll
9/11/2002 3:11:09 PM 12 Interop.TASKSCHEDULERLib.dll
9/11/2002 3:11:10 PM 56 Interop.VBIDE.dll
[1]$ dir | format/table | convert/csv | out/console
"Written","Length.kb","Name"
"6/28/2002 12:52:10 PM","189","sd.exe"
"11/7/2002 10:59:18 AM","0","sd.ini"
"6/28/2002 12:52:10 PM","2633","sd.pdb"
"6/28/2002 12:52:18 PM","223","sdscc.dll"

Фиг.23