ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ

УДК 681.324

С. А. Зинкин

ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ СЕТЯМИ ХРАНЕНИЯ И ОБРАБОТКИ ДАННЫХ НА ОСНОВЕ НЕПОСРЕДСТВЕННОЙ ИНТЕРПРЕТАЦИИ ЛОГИКО-АЛГЕБРАИЧЕСКИХ СПЕЦИФИКАЦИЙ

Аннотация. Рассмотрены основы синтеза управляющих распределенных программ на основе формальных логико-алгебраических спецификаций. Приведены иллюстрирующие примеры, развитые на основе модели использования ресурсов Генриха – Глэссера. Даны логико-алгебраические спецификации для реализации нового подхода к реализации распределенных сетевых программ.

Ключевые слова: хранение данных, обработка данных, формальное определение распределенных процессов, логико-алгебраический подход, сети абстрактных машин, распределенные поведенческие модели.

Abstract. The article describes the basics of a synthesis of distributed control programs based on formal logical-algebraic specifications. The author gives examples illustrating the developed Genrich-Glaesser model and introduces the logical-algebraic specifications for a new approach to implementation of distributed network applications.

Key words: data storage, data processing, formal representations of distributed processes, logical-algebraic approach, networks of abstract machines, distributed behavioral models.

Введение

Рассмотрим вопросы построения формальных сетевых спецификаций, описывающих распределение единиц некоторого ресурса *R* между запросами различных типов. Детализированные спецификации должны обеспечить эффективную реализацию сети программных модулей, управляющих распределенным сетевым ресурсом, например, реплицированными базами данных в предположении, что каждая копия базы данных размещена на собственном внешнем запоминающем устройстве (ВЗУ). На рис. 1 представлена типовая структура составной сети с распределенным ресурсом *R*. Здесь буквами *M* и *K* обозначены маршрутизаторы глобальной сети и коммутаторы локальных сетей соответственно, черными кружками обозначены узлы-единицы ресурса *R*, на которых установлены системы управления базами данных и к которым подключены обозначенные цилиндрами ВЗУ для хранения информации. Затемненными кружками представлены узлы-источники запросов на использование распределенного ресурса *R*.

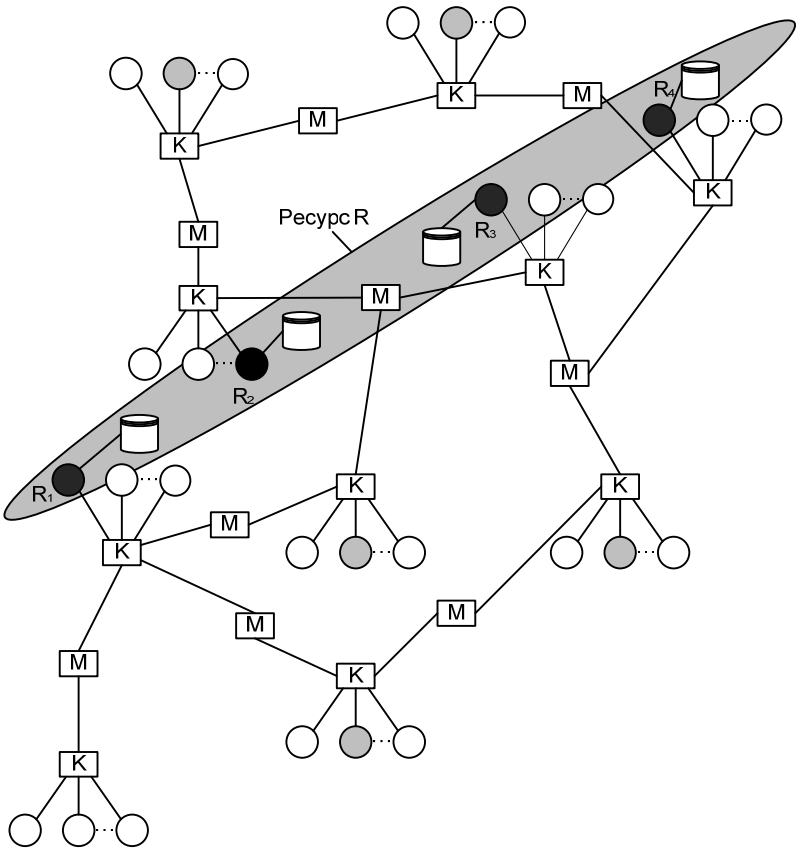

Рис. 1. Реализация распределенного ресурса в вычислительной сети

В основу описания функционирования подобной системы положим показательный пример, предложенный в работах [1–3]. В работах [1, 2] модель использования ресурсов *R* для запросов двух типов иллюстрирует предложенные автором сети «предикат–переход», а в работе [3] та же модель описана сетями машин абстрактных состояний. Недостатком рассмотренных моделей является то, что без использования приоритетных механизмов между запросами двух типов время обслуживания *e*-запросов резко увеличивается с ростом числа *s*-запросов и система становится практически неработоспособной. Кроме того, в перечисленных работах не формализована процедурная составляющая модели. Предложенный в работах [4, 5] формализм сетей абстрактных машин (СеАМ) и его расширение (РСеАМ) и реализации данных сетей на базе FS-технологии, описанной в работах [6, 7], в б*о*льшей степени учитывают механизмы непосредственно интерпретируемых спецификаций, обеспечивают иерархическое проектирование распределенных программ. СеАМ-выражения строятся на базе суперпозиций операторов α-дизъюнкций, а в выражениях для расширенного варианта РСеАМ дополнительно используются суперпозиции операторов α-итераций из алгебры алгоритмов [8–10], хотя для записи условных и циклических выражений могла быть выбрана и другая подходящая нотация, например из [11].

Условиями в выражениях для модулей СеАМ и РСеАМ называются любые логические формулы с предикатными символами, используемыми в качестве логических переменных. В выражениях могут использоваться кванторы, функциональные символы с предикатом равенства, а сами выражения могут составляться на основе использования логик первого и высших порядков. Кроме того, в качестве логических переменных здесь могут быть использованы квантифицированные операторы выбора $\tilde{\exists}!$, $\tilde{\exists}!!$, $\tilde{\forall}$ и $\tilde{\forall}!!$, к которым применен оператор «подчеркивание» [4, 5]. При выполнении оператора $\tilde{\exists}!$ из области истинности предиката выбирается произвольный кортеж. При выполнении оператора $\tilde{\exists}!!$ выбирается единственный кортеж, находящийся в области истинности предиката. При выполнении оператора $\tilde{\forall}$ выбираются все кортежи, составляющие область истинности соответствующего предиката. Оператор $\tilde{\forall}!!$ позволяет выбрать все кортежи из области истинности предиката в случае, если его область определения совпадает с его же областью истинности. Во всех случаях подразумевается, что предикат описывается выражением, стоящим справа от символа квантифицированного оператора. Каждому из описанных квантифицированных операторов выборки кортежей из отношений, в случае его использования в условной части выражения для модуля, ставится в соответствие элементарное логическое условие, истинное в случае успешного выполнения оператора и ложное в противном случае.

Целью настоящей работы является построение сети абстрактных машин, управляющей распределенным сетевым ресурсом хранения R. Детализированные модели функционирования распределенного ресурса R, построенные на основе формализма СеАМ, представлены в работах [12, 13].

1. Построение управляющей сети абстрактных машин

Рассмотрим сеть абстрактных машин, содержащую каузально связанные модули, осуществляющие управление распределенным ресурсом R. Пусть $A = \{a_1, a_2, \ldots, a_{|A|}\}$ – множество запросов к модулям ВЗУ, подключенным к сети и представляющим в совокупности распределенный ресурс, состоящий из множества единиц R (и содержащим искомые копии одной и той же базы данных); T – множество «жетонов», или «билетов», дающих возможность получить доступ к единицам ресурса R. Число жетонов $n_T = |T|$ совпадает с числом единиц $n_R = |R|$ ресурса R. Абстрактная структура моделируемой системы представлена на рис. 2.

На рис. 2 прямоугольниками представлены очереди, представленные в модели унарными предикатами, причем имена очередей использованы далее в качестве индексов соответствующих предикатных имен. Кружками представлены множества запросов, жетонов и единиц сетевого ресурса.

СеАМ-выражением для модуля m_0 описывается установка режима обслуживания запроса и его перевод из очереди p_{got} в очередь p_w:

$$m_0 = [(\tilde{\exists}!x \in A)p_{got}(x)]([(\tilde{\exists}!m \in M)p_{reg}(m)]$$

$$(\{p_{got}(x) \leftarrow \text{false},\ f_m(x) \leftarrow m,\ p_w(x) \leftarrow \text{true}\} \vee R^E) \vee R^E). \qquad (1)$$

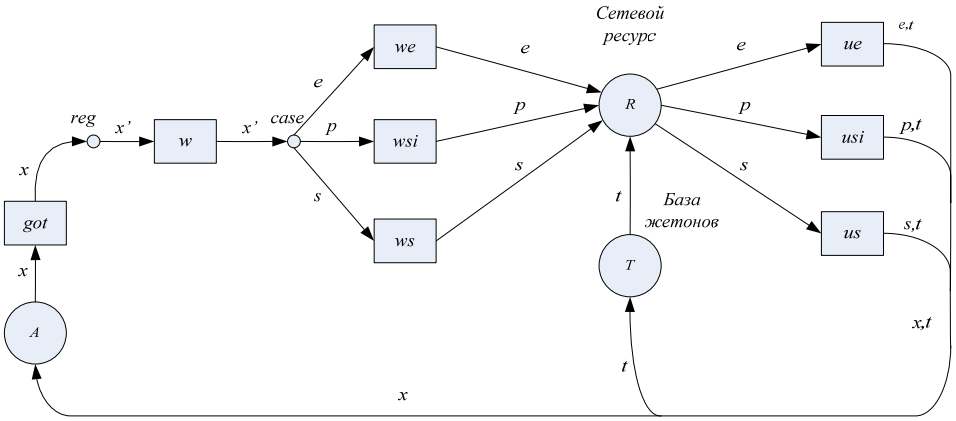

Рис. 2. Абстрактная структура сети хранения данных

В выражении для модуля m_0 используются следующие предикаты и одна функция:

$p_{got}(x)$ – унарный предикат, определяющий готовность клиента сети ВЗУ к выдаче запроса $x \in A$;

$p_{reg}(m)$ – унарный предикат, задающий один из трех режимов обработки запросов, $m \in M$, $M = \{e, p, s\}$: e – запрос, требующий для своего выполнения всех единиц ресурса R, представленного совокупностью модулей ВЗУ на магнитных дисках; p – запрос, требующий определенную единицу ресурса R, т.е. адресуемый модуль ВЗУ; s – запрос, для выполнения которого достаточно любой единицы ресурса R;

$f_m(x)$ – функция, используемая для указания на режим или тип обработки запроса x;

$p_w(x)$ – унарный предикат, область истинности которого представляет общую очередь запросов (без различия по типу запросов).

В дальнейшем там, где это не противоречит контексту, мы будем использовать обычное для имитационного моделирования отождествление функций, предикатов или отношений с моделируемыми объектами: например, унарный предикатный символ p_{got} будем использовать и для именования очереди клиентских запросов, а унарным предикатным символом p_w обозначим очередь запросов, для каждого из которых определен режим обслуживания. Таким образом, СеАМ-выражением для модуля m_0 описывается установка режима обслуживания каждого запроса и перевод данного запроса из очереди p_{got} в очередь p_w.

В связи с тем, что условия в квадратных скобках не зависят друг от друга, выражение (1) можно представить и в следующем семантически эквивалентном виде:

$$m_0 = [\underline{((\tilde{\exists}!x \in A)\, p_{got}(x))} \,\&\, \underline{((\tilde{\exists}!m \in M)\, p_{reg}(m))}]$$

$$(\{p_{got}(x) \leftarrow \text{false},\ f_m(x) \leftarrow m,\ p_w(x) \leftarrow \text{true}\} \vee R^E). \qquad (2)$$

В результате выполнения квантифицированных операторов $\tilde{\exists}!$ и $\tilde{\forall}$ формируются истинные или ложные значения условий успешного или не-

успешного их выполнения. Эти условия формируются путем применения оператора «подчеркивание», который при единственности условия в квадратных скобках может опускаться.

Модуль m_e осуществляет выбор запросов e-типа из очереди p_w и формирует из выбранных запросов очередь p_{we}:

$$m_e = [(\tilde{\exists}!x \in A)(p_w(x) \,\&\, (f_m(x) = e))]$$

$$(\{p_w(x) \leftarrow \text{false},\ p_{we}(x) \leftarrow \text{true}\} \vee R^E). \qquad (3)$$

В выражении для модуля m_e используются унарные предикаты $p_w(x)$ и $p_{we}(x)$, а также функция $p_m(x)$.

Модуль m_{sif} выбирает из очереди p_w запросы p-типа и помещает их в очередь p_{wsif}:

$$m_{sif} = [(\tilde{\exists}!x \in A)(p_w(x) \,\&\, (f_m(x) = p))]$$

$$(\{p_w(x) \leftarrow \text{false},\ p_{wsif}(x) \leftarrow \text{true}\} \vee R^E). \qquad (4)$$

При выполнении модуля m_{si} запрос p-типа переходит из очереди p_{wsif} в очередь p_{wsi} со сформированным номером (адресом) единицы ресурса R:

$$m_{si} = [(\tilde{\exists}!x \in A)p_{wsif}(x)][(\tilde{\exists}!i \in I)p_{irand}(i)]$$

$$(\{f_{num}(x) \leftarrow i,\ p_{wsif}(x) \leftarrow \text{false},\ p_{wsi}(x) \leftarrow \text{true}\} \vee R^E) \vee R^E). \qquad (5)$$

Модуль m_{si} для запросов p-типа формирует номер i единицы ресурса R с помощью оператора $(\tilde{\exists}!i \in I)p_{irand}(i)$ путем псевдослучайного равновероятного выбора одного из значений переменной i ($i = 1, 2, \ldots, n_R$); здесь $p_{irand}(i)$ – унарный предикат; $f_{num}(x)$ – унарная функция для сохранения выбранного значения переменной i; $p_{wsif}(x)$ и $p_{wsi}(x)$ – унарные предикаты, области истинности которых задают соответственно одноименные очереди p_{wsif} и p_{wsi}; n_R – число единиц ресурса R.

Выражение (5) представимо также и в следующем семантически эквивалентном виде:

$$m_{si} = [\underline{((\tilde{\exists}!x \in A)p_{wsif}(x))} \,\&\, \underline{((\tilde{\exists}!i \in I)p_{irand}(i))}]$$

$$(\{f_{num}(x) \leftarrow i,\ p_{wsif}(x) \leftarrow \text{false},\ p_{wsi}(x) \leftarrow \text{true}\} \vee R^E). \qquad (6)$$

Модуль m_s выбирает из очереди $p_w(x)$ запросы s-типа и помещает их в очередь $p_{ws}(x)$:

$$m_s = [(\tilde{\exists}!x \in A)(p_w(x) \,\&\, (f_m(x) = s))]$$

$$(\{p_w(x) \leftarrow \text{false},\ p_{ws}(x) \leftarrow \text{true}\} \vee R^E). \qquad (7)$$

Модуль m_{ate} выполняет необходимые проверки для дальнейшего выполнения запроса e-типа:

$$m_{ate} = [(\tilde{\exists}!x \in A)p_{we}(x)][(\forall t \in T)(\neg p_{work}(t))]$$

$$([(\tilde{\exists}!q \in \{r, w\})p_{tip}(q)]([(\tilde{\forall}t \in T)p_{av}(t)]$$

$$(\{p_{ate}(x,t) \leftarrow \text{true},\ p_{av}(t) \leftarrow \text{false},\ f_{oper}(t) \leftarrow q,\ p_{work}(t) \leftarrow \text{true},$$

$$p_{we}(x) \leftarrow \text{false},\ p_{ue}(x) \leftarrow \text{true}\} \vee R^E) \vee R^E) \vee R^E) \vee R^E). \qquad (8)$$

Для обслуживания одного подобного запроса необходимо иметь все жетоны $t \in T$. Оператор $(\tilde{\exists}!x \in A)p_{we}(x)$ выбирает один из запросов *e*-типа из очереди p_{we}. Затем проверяется истинность выражения $(\forall t \in T)(\neg p_{work}(t))$ – условия незанятости рабочей операцией записи или чтения данных всех модулей ВЗУ. Оператор $(\tilde{\exists}!q \in \{r, w\})p_{tip}(q)$ выбирает тип операции – чтения (*r*) или записи (*w*) данных. Оператор $(\tilde{\forall}t \in T)p_{av}(t)$ выбирает все жетоны из области истинности предиката $p_{av}(t)$, совпадающей при $(\forall t \in T)(\neg p_{work}(t)) = \text{true}$ с областью его определения. Предикат $p_{av}(t)$ характеризует наличие (при $p_{av}(t_i)$ = true) или отсутствие (при $p_{av}(t_i)$ = false) свободного жетона t_i, позволяющего занять *i*-ю единицу ресурса *R*. Напомним, что в результате выполнения квантифицированных операторов $\tilde{\exists}!$ и $\tilde{\forall}$ формируются истинные или ложные значения условий успешного или неуспешного их выполнения. Если все проверки оказались успешными, то далее выбранный запрос (представленный значением предметной переменной *x*) связывается со всеми жетонами $t \in T$ с помощью правила обновления предиката $p_{ate}(x,t) \leftarrow \text{true}$. Пусть в данном случае для некоторого запроса a_j *e*-типа с помощью оператора $(\tilde{\forall}t \in T)p_{av}(t)$ и при $(\forall t \in T)(\neg p_{work}(t)) = \text{true}$ были выбраны все жетоны $t \in T$. Поэтому в соответствии с правилом $p_{ate}(x,t) \leftarrow \text{true}$ в блоке модуля m_{ate} будет реально выполнено $n_T = |T|$ обновлений предиката p_{ate}:

$$p_{ate}(a_j, t_1) \leftarrow \text{true},\ p_{ate}(a_j, t_2) \leftarrow \text{true},\ \ldots,\ p_{ate}(a_j, t_{|T|}) \leftarrow \text{true},$$

т.е. будет сформировано отношение

$$\{< a_j, t_1 >, < a_j, t_2 >, \ldots, < a_j, t_{|T|} >\},$$

связывающее выбранный запрос a_j со всеми жетонами множества $T = \{t_1, t_2, \ldots, t_{|T|}\}$. Вместо оператора выбора $(\tilde{\forall}t \in T)p_{av}(t)$ можно было бы использовать оператор $(\tilde{\forall}!!t \in T)p_{av}(t)$, но в данном случае в этом нет необходимости, так как при истинном условии $(\forall t \in T)(\neg p_{work}(t))$ все жетоны были свободны. Далее выполняется n_T обновлений предиката $p_{av}(t)$, в результате чего все жетоны становятся занятыми; в результате n_T обновлений унарной функции $f_{oper}(t)$ задается «коллективная» операция записи или чтения данных; n_T обновлений унарного предиката $p_{work}(t)$ инициируют рабочие операции для каждого из модулей ВЗУ (затрагивающие, естественно, и другие устройства системы ВЗУ). Далее запрос перемещается из очереди p_{we} в очередь p_{ue}, в которой он остается на все время выполнения коллективной операции в системе ВЗУ.

При выполнении модуля m_{ati} запрос *p*-типа передается из очереди p_{wsi} в очередь p_{usi} при условии наличия соответствующего жетона с требуемым номером:

$$m_{ati} = [(\tilde{\exists}!x \in A)p_{wsi}(x)]$$

$$([[(\underline{(\tilde{\exists}!t \in T)(p_{av}(t) \,\&\, (f_{num}(x) = f_{ind}(t)))} \,\&\, \neg((\exists z \in A)p_{we}(z))]$$

$$([\neg p_{work}(t)][(\tilde{\exists}!q \in \{r, w\})p_{tip}(q)](\{f_{ati}(x) \leftarrow t,\ p_{av}(t) \leftarrow \text{false},$$

$$p_{wsi}(x) \leftarrow \text{false},\ p_{usi}(x) \leftarrow \text{true},\ f_{oper}(t) \leftarrow q,\ p_{work}(t) \leftarrow \text{true}\}$$

$$\vee R^E) \vee R^E) \vee R^E) \vee R^E). \qquad (9)$$

Здесь для нумерации жетонов используется функция $f_{ind}(t)$. Данный модуль выполняется при условии, что в очереди p_{we} отсутствуют запросы *e*-типа, обладающие более высоким относительным приоритетом, чем запросы *p*-типа. Данное условие задается выражением $\neg((\exists z \in A)p_{we}(z))$, принимающим истинное значение при пустой области истинности предиката $p_{we}(z)$. В остальном модуль m_{ati} выполняется аналогично модулю m_{ate} с инициализацией одной рабочей операции записи или чтения данных в отдельном модуле ВЗУ, номер которого, как единицы ресурса R, задается функцией $f_{num}(x)$. Функция $f_{ati}(x)$ связывает выбранный жетон с выбранным ранее запросом, для чего выполняется обновление функции $f_{ati}(x) \leftarrow t$. В дальнейшем этот факт связи будет использован для корректного возвращения жетона модулем m_{usi}.

Модуль m_{ta} выбирает запрос *s*-типа из очереди p_{ws} и передает его в очередь p_{us}, в которой он будет пребывать до полного окончания выполнения рабочей операции:

$$m_{ta} = [(\tilde{\exists}!x \in A)p_{ws}(x)]$$

$$([[(\underline{(\tilde{\exists}!t \in T)p_{av}(t))} \,\&\, \neg((\exists w \in A)p_{we}(w)) \,\&\, \neg((\exists z \in A)p_{wsi}(z))]$$

$$([\neg p_{work}(t)][(\tilde{\exists}!q \in \{r, w\})p_{tip}(q)](\{f_{ta}(t) \leftarrow x,\ p_{av}(t) \leftarrow \text{false},$$

$$p_{ws}(x) \leftarrow \text{false},\ p_{us}(x) \leftarrow \text{true},\ f_{oper}(t) \leftarrow q,\ p_{work}(t) \leftarrow \text{true}\} \vee$$

$$\vee R^E) \vee R^E) \vee R^E) \vee R^E). \qquad (10)$$

Запросу *s*-типа для выполнения нужен один произвольно выбранный жетон t; для связывания жетона t с запросом x используется функция $f_{ta}(t)$, обновляемая с помощью правила $f_{ta}(t) \leftarrow x$. Данная связь в дальнейшем используется модулем m_{us} для корректного возвращения жетона. Запрос *s*-типа обладает самым низким относительным приоритетом, поэтому он выполняется при условии, что в очередях p_{we} и p_{wsi} отсутствуют запросы *e*-типа и *p*-типа. Это условие задается формулой $\neg((\exists w \in A)p_{we}(w)) \,\&\, \neg((\exists z \in A)p_{wsi}(z))$, включенной в α-условие модуля m_{ta}. В остальном модуль выполняет действия, аналогичные действиям модулей m_{ate} и m_{ati}.

Модуль m_{ue} фиксирует окончание обслуживания *e*-запроса, а затем он удаляет данный запрос из очереди p_{ue}, если истинно условие $\neg((\exists t \in T)p_{work}(t))$, что свидетельствует об окончании выполнения коллективной рабочей операции во всех модулях ВЗУ:

$$m_{ue} = [\underline{((\tilde{\exists}!x \in A) p_{ue}(x))} \,\&\, \neg((\exists t \in T) p_{work}(t))]$$

$$([(\tilde{\forall} t \in T) f_T(t)](\{p_{ue}(x) \leftarrow \text{false}, p_{got}(x) \leftarrow \text{true}, p_{av}(t) \leftarrow \text{true},$$

$$f_{oper}(t) \leftarrow \text{undef}, p_{ate}(x,t) \leftarrow \text{false}\} \vee R^E) \vee R^E). \qquad (11)$$

Оператор $(\tilde{\forall} t \in T) p_T(t)$ выбирает все жетоны, представленные в модели значениями предметной переменной t в области истинности характеристической функции (унарного предиката) p_T. Далее разрываются все связи e-запроса с жетонами путем обновления предиката $p_{ate}(x,t)$, а сам запрос после удаления из очереди p_{ue} возвращается во входную очередь p_{got}. Все жетоны $t \in T$ освобождаются при выполнении правила $p_{av}(t) \leftarrow$ true.

Модуль m_{usi} возвращает запрос p-типа во входную очередь p_{got} после завершения выполнения рабочей операции определенным модулем ВЗУ:

$$m_{usi} = [(\tilde{\exists}!x \in A) p_{usi}(x)]([(\tilde{\exists}!t \in T)(f_{ati}(x) = t)]([\neg p_{work}(t)]$$

$$(\{p_{usi}(x) \leftarrow \text{false}, p_{got}(x) \leftarrow \text{true}, p_{av}(t) \leftarrow \text{true}, f_{ati}(x) \leftarrow \text{undef},$$

$$f_{oper}(t) \leftarrow \text{undef}, f_{num}(x) \leftarrow \text{undef}\} \vee R^E) \vee R^E) \vee R^E). \qquad (12)$$

После выполнения последовательности вычислений и проверок трех α-условий p-запрос удаляется из очереди p_{usi} и возвращается во входную очередь p_{got}. Использованный жетон освобождается ($p_{av}(t) \leftarrow$ true) и его связь с запросом аннулируется ($f_{ati}(x) \leftarrow$ undef).

Модуль m_{us} выполняется аналогично предыдущему модулю m_{usi}; он фиксирует окончание обслуживания s-запроса, возвращает его во входную очередь p_{got}, а также возвращает использованный жетон:

$$m_{us} = [(\tilde{\exists}!x \in A) p_{us}(x)]([(\tilde{\exists}!t \in T)(f_{ta}(t) = x)]([\neg p_{work}(t)]$$

$$(\{p_{us}(x) \leftarrow \text{false}, p_{got}(x) \leftarrow \text{true}, p_{av}(t) \leftarrow \text{true},$$

$$f_{oper}(t) \leftarrow \text{undef}, f_{ta}(t) \leftarrow \text{undef}\} \vee R^E) \vee R^E) \vee R^E). \qquad (13)$$

Для запуска сети необходимо назначить каждому модулю по агенту-серверу. Агенты-серверы циклически опрашивают α-условия модулей и определяют порядок выполнения запросов. Анализ предметной области и особенности FS-технологии показывает, что модулям m_0, m_{sif}, m_{si}, m_{usi}, m_s и m_{us} может быть назначено по $n_A = |A|$ агентов-серверов, модулям m_{ati} и m_{ta} – по $n_T = |T|$ агентов-серверов, а модулям m_e, m_{ate} и m_{ue} достаточно назначить по одному агенту. Одиннадцать модулей СеАМ взаимодействуют через разделяемое FS-пространство функций и предикатов; часть модулей, связанных с обработкой запросов p- и s-типов, могут выполняться несколькими агентами-серверами, или «мультиагентами-серверами». При реализации каузальных связей модули взаимодействуют, модифицируя и проверяя функции и предикаты (точнее, информационные объекты, представляющие указанные функции и предикаты в FS-пространстве), находящиеся в ограниченных окрестно-

стях FS-пространства, что может быть использовано для более эффективного размещения данных объектов и повышения эффективности функционирования реализации сети СеАМ в целом.

Основным преимуществом сетей СеАМ является то, что запуск модулей на выполнение реализуется естественно, по мере готовности, или истинности логических условий. При этом ряд модулей, размещенных на различных физических узлах вычислительной сети, может выполняться параллельно (естественный параллелизм). Другие модули могут выполняться, конкурируя из-за ресурсов – функций и предикатов, составляющих FS-пространство. Взаимодействие модулей, каузально связанных посредством FS-пространства, иллюстрирует рис. 3.

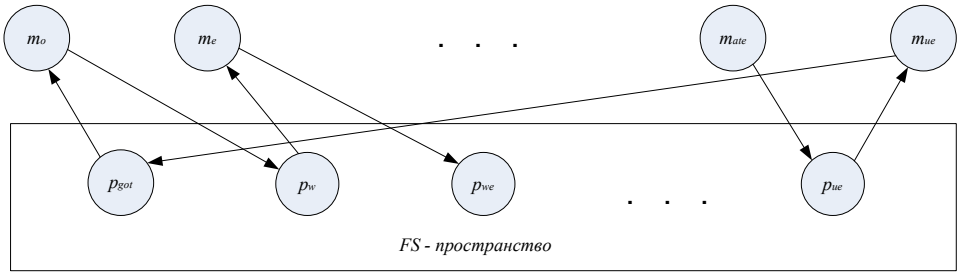

Рис. 3. Организация каузальных межмодульных связей

Здесь стрелки, ведущие от предикатов к модулям, обозначают факты проверки предикатов модулями, а стрелки, ведущие от модулей к предикатам, обозначают факты модификации предикатов модулями.

2. Предотвращение тупиковых ситуаций в сети машин абстрактных состояний при формировании сложных запросов

Формализм СеАМ позволяет осуществлять модификации описаний дискретных систем. В качестве примера рассмотрим систему организации доступа к элементам (единицам) распределенного ресурса, описанную выражениями (1)–(13). К запросам вида *e* (использование всех единиц ресурса), *p* (использование предопределенной, заранее проиндексированной единицы ресурса *R*), *s* (использование любой свободной единицы ресурса) добавим новый тип запроса, например, *k*-запрос. В отличие от запроса типа *e*, запрос типа *k* собирает жетоны на право использования единиц ресурса по одному до тех пор, пока у него не окажется полный набор жетонов. Затем для запроса типа *k* выполняется некоторая групповая операция, требующая наличия всех единиц ресурса *R*. Напомним, что запрос типа *e* также требует наличия всех свободных единиц ресурса до начала групповой обработки.

Дополнительные модули m_{ei}, m_{atei}, m_{atall}, m_{uei}, описываемые приведенными ниже выражениями (14)–(17), позволяют реализовать в модели обслуживание запроса типа *k* при условии, что запрос типа *e* обладает более высоким относительным приоритетом по сравнению с *k*-запросом (без прерывания начавшегося обслуживания). Этот приоритет реализуется путем реализации проверки условия $\neg(\exists z \in A) p_{we}(z)$ в выражении (15). В свою очередь,

k-запрос должен обладать более высоким относительным приоритетом по сравнению с запросами *p* и *s*; для реализации такого приоритета необходимо в условной части модулей (9) и (10) реализовать проверку условия $\neg(\underline{\tilde{\exists}!(x \in A,\ t \in \mathrm{T})\, p_{wei}(x,t)})$.

Модуль m_{ei} формирует очередь p_{wei}, выбирая *k*-запрос из очереди p_w:

$$m_{ei} = [(\tilde{\exists}! x \in A)(p_w(x)\ \&\ (f_m(x) = k))]([\neg(\underline{\tilde{\exists}!(x \in A,\ t \in \mathrm{T})\, p_{wei}(x,t)})]$$

$$([(\tilde{\forall} t \in T)\, p_T(t)](\{p_w(x) \leftarrow \text{false},\ p_{wei}(x,t) \leftarrow \text{true}\} \vee R^E) \vee R^E) \vee R^E) \vee R^E). \quad (14)$$

Здесь связь запроса *x* с жетоном *t* в очереди p_{wei} является «фиктивной» – она лишь «декларирует намерение» *k*-запроса получить жетоны в будущем. В очереди p_{wei} может находиться не более одного *k*-запроса.

Модуль m_{atei} выбирает освобождающиеся жетоны по одному и формирует новую очередь p_{atw}, в которой устанавливаются реальные связи выбранного *k*-запроса с выбираемыми жетонами:

$$m_{atei} = [\neg(\exists z \in A)\, p_{we}(z)]([(\tilde{\forall}!! t \in T)\, p_{av}(t)]([\tilde{\exists}!(x \in A)\, p_{wei}(x,t)]$$

$$(\{p_{atw}(x,t) \leftarrow \text{true},\ p_{av}(t) \leftarrow \text{false},\ p_{wei}(x,t) \leftarrow \text{false}\} \vee R^E) \vee R^E) \vee R^E). \quad (15)$$

Добавление в сеть модуля m_{atei} может привести к тупиковой ситуации в системе, так как при поступлении в очередь p_{we} *e*-запроса более высокого приоритета сбор жетонов для *k*-запроса будет блокирован.

Остальные выражения для дополнительных модулей новой сети имеют следующий вид. Модуль m_{atall} переводит *k*-запрос из очереди p_{atw} в очередь p_{uei} после сбора всех жетонов. Во время пребывания *k*-запроса в этой очереди выполняется рабочая операция записи или чтения всеми модулями ВЗУ:

$$m_{atall} = [(\forall t \in T) \neg p_{av}(t)]([\tilde{\forall}(x \in A, t \in T)\, p_{atw}(x,t)]$$

$$([(\tilde{\exists}! q \in \{r,w\})\, p_{tip}(q)]([(\forall t \in T) \neg p_{work}(t)]$$

$$(\{p_{atw}(x,t) \leftarrow \text{false},\ f_{oper}(t) \leftarrow q,\ p_{work}(t) \leftarrow \text{true},$$

$$p_{uei}(x) \leftarrow \text{true}\} \vee R^E) \vee R^E) \vee R^E) \vee R^E). \quad (16)$$

Модуль m_{uei} возвращает все использованные жетоны и перемещает *k*-запрос из очереди p_{uei} во входную очередь по завершении обслуживания:

$$m_{uei} = [\underline{((\tilde{\exists}! x \in A)\, p_{uei}(x))}\ \&\ \neg((\exists t \in T)\, p_{work}(t))]([(\tilde{\forall} t \in T)\, p_T(t)]$$

$$(\{p_{uei}(x) \leftarrow \text{false},\ p_{got}(x) \leftarrow \text{true},\ p_{av}(t) \leftarrow \text{true},$$

$$f_{oper}(t) \leftarrow \text{undef}\} \vee R^E) \vee R^E). \quad (17)$$

Модуль m'_{ati} заменяет в новой сети модуль m_{ati} и отличается от него тем, что в его условную часть добавлена проверка условия отсутствия *k*-запросов в очереди p_{wei}:

$$m'_{ati} = [(\tilde{\exists}!x \in A)p_{wsi}(x)]([\neg(\underline{\tilde{\exists}!(x \in A,\ t \in \mathrm{T})p_{wei}(x,t)})]$$

$$([(\underline{(\tilde{\exists}!t \in T)(p_{av}(t)\ \&\ (f_{num}(x) = f_{ind}(t)))}\ \&\ \neg((\exists z \in A)p_{we}(z))]$$

$$([\neg p_{work}(t)]([(\tilde{\exists}!q \in \{r,w\})p_{tip}(q)]$$

$$(\{f_{ati}(x) \leftarrow t,\ p_{av}(t) \leftarrow \text{false},\ p_{wsi}(x) \leftarrow \text{false},\ p_{usi}(x) \leftarrow \text{true},$$

$$f_{oper}(t) \leftarrow q,\ p_{work}(t) \leftarrow \text{true}\} \vee R^E) \vee R^E) \vee R^E) \vee R^E). \qquad (18)$$

Модуль m'_{ta} заменяет в новой сети модуль m_{ta} и отличается от него, как и предыдущий модуль, наличием в его условной части проверки условия отсутствия k-запросов в очереди p_{wei}:

$$m'_{ta} = [(\tilde{\exists}!x \in A)p_{ws}(x)]([\neg(\underline{\tilde{\exists}!(x \in A,\ t \in \mathrm{T})p_{wei}(x,t)})]$$

$$([((\tilde{\exists}!t \in T)p_{av}(t)) \& \neg((\exists w \in A)p_{we}(w)) \& \neg((\exists z \in A)p_{wsi}(z))]$$

$$([\neg p_{work}(t)]([(\tilde{\exists}!q \in \{r,w\})p_{tip}(q)]$$

$$(\{f_{ta}(t) \leftarrow x,\ p_{av}(t) \leftarrow \text{false},\ p_{ws}(x) \leftarrow \text{false},\ p_{us}(x) \leftarrow \text{true},$$

$$f_{oper}(t) \leftarrow q,\ p_{work}(t) \leftarrow \text{true}\} \vee R^E) \vee R^E) \vee R^E) \vee R^E). \qquad (19)$$

Для разрешения тупиковой ситуации без изменения системы приоритетов предлагается следующий способ: при поступлении некоторого запроса e-типа, отмечаемого обновлением предиката $p_{we}(a_i) \leftarrow \text{true}$, запрос k-типа «отдает» захваченные ранее жетоны по одному с помощью модулей m_{atoff} и m'_{atoff}:

$$m_{atoff} = [(\exists x \in A)p_{we}(x)]([\tilde{\exists}!(x \in A, t \in T)p_{atw}(x,t)]$$

$$(\{p_{atw}(x,t) \leftarrow \text{false},\ p_{ei}(x) \leftarrow \text{true}\} \vee R^E) \vee R^E); \qquad (20)$$

$$m'_{atoff} = [\tilde{\exists}!(x \in A, t \in T)p_{wei}(x,t)]([p_{ei}(x)]$$

$$(\{p_{wei}(x,t) \leftarrow \text{false},\ p_{ei}(x) \leftarrow \text{false},\ p_{av}(t) \leftarrow \text{true}\} \vee R^E) \vee R^E). \qquad (21)$$

Другим способом недопущения тупиковой ситуации является обеспечение одинаковых приоритетов для e- и k-запросов; в этом случае условная часть выражения (15) не должна содержать проверку условия $\neg(\exists z \in A)p_{we}(z)$, а модули m_{atoff} и m'_{atoff} при этом не нужны:

$$m'_{atei} = [(\tilde{\exists}!t \in T)p_{av}(t)]([\tilde{\exists}!(x \in A)p_{wei}(x,t)]$$

$$(\{p_{atw}(x,t) \leftarrow \text{true},\ p_{av}(t) \leftarrow \text{false},\ p_{wei}(x,t) \leftarrow \text{false}\} \vee R^E) \vee R^E). \qquad (22)$$

Тупиковая ситуация не возникнет также, если запросу *k*-типа присвоить относительный приоритет над запросами типа *e*; при этом условная часть выражения (15) не должна содержать проверку условия $\neg(\exists z \in A) f_{we}(z)$, а в условную часть выражения (8) необходимо добавить проверку высказывания $\neg(\underline{\tilde{\exists}!(x \in A,\ t \in \mathrm{T}) p_{wei}(x,t)})$:

$$m'_{ate} = [(\tilde{\exists}! x \in A) p_{we}(x)]([(\forall t \in T)(\neg p_{work}(t))]$$

$$([(\tilde{\exists}! q \in \{r, w\}) p_{tip}(q)]([(\forall t \in T) p_{av}(t)]([(\tilde{\forall} t \in T) p_T(t)]$$

$$[\neg(\underline{\tilde{\exists}!(x \in A,\ t \in \mathrm{T}) p_{wei}(x,t)})]$$

$$(\{p_{ate}(x,t) \leftarrow \text{true},\ p_{av}(t) \leftarrow \text{false},\ f_{oper}(t) \leftarrow q,$$

$$p_{work}(t) \leftarrow \text{true},\ p_{we}(x) \leftarrow \text{false},\ p_{ue}(x) \leftarrow \text{true}\} \vee$$

$$\vee R^E) \vee R^E) \vee R^E) \vee R^E) \vee R^E) \vee R^E). \qquad (23)$$

Следует учитывать, что два последних варианта недопущения тупиковой ситуации изменяют первоначально поставленные условия. Для недопущения тупиковой ситуации можно также просто выполнять запросы *e*-типа в режиме выполнения запросов *k*-типа. Выбор приемлемой реализации приоритетных механизмов в сети ВЗУ может быть осуществлен с помощью статистических экспериментов с имитационными моделями, реализованными на основе формализма временны*ы*х СеАМ.

3. Использование расширенного варианта формализма для описания процессов выбора и обработки сложных запросов в вычислительной сети с копиями базы данных

Опишем формирование и управление выполнением сложных запросов к системе ВЗУ с помощью расширенного варианта формализма РСеАМ. Формализм РСеАМ отличается от формализма СеАМ «усиленной» за счет использования темпоральных, или временных, операций процедурной составляющей модели представления знаний. В приведенных ниже выражениях используются две темпоральные операции – «непосредственного следования» и «возможно одновременного выполнения» действий, обозначенные упрощенно символами «;» и «,» соответственно. Заметим, что в выражениях для модулей заключительные символы «;» являются обычными разделительными межмодульными символами и их следует отличать от упрощенного обозначения операции «непосредственного следования». Наряду с операцией «α-дизъюнкция» здесь разрешено использование операции «α-итерация». Выражения (24)–(27) описывают выполнение запросов *e*, *p* и *s* типов в системе или сети ВЗУ. Модуль РСеАМ состоит из основного модуля m_{esis} и дополнительных фрагментов m_A, m_B и m_C, описывающих обработку запросов типов *e*, *p* и *s* соответственно:

$$m_{esis} = [\neg p_s(\alpha)]\{[(\tilde{\exists}! x \in A) p_{got}(x)]\{R^E\};\ (\tilde{\exists}! m \in M) p_{reg}(m);$$

$$(p_{got}(x) \leftarrow \text{false},\ f_m(x) \leftarrow m,\ p_w(x) \leftarrow \text{true});$$

$$[f_m(x)=e](m_A \vee [f_m(x)=p](m_B \vee [f_m(x)=s](m_C \vee R^E)))\}, \qquad (24)$$

где p_s – стартовый предикат, α – константа; при $p_s(\alpha)$ = true модуль начинает, а при $p_s(\alpha)$ = false заканчивает свою работу;

$$m_A = ((p_w(x) \leftarrow \text{false},\ p_{we}(x) \leftarrow \text{true});\ [(\forall t \in T)p_{av}^{+}(t)]\{R^E\};$$

$$(\tilde{\forall} t \in T)p_T(t)(p_{ate}(x,t) \leftarrow \text{true},\ p_{av}(t) \leftarrow \text{false});\ \ p_{av}^{-};$$

$$(p_{we}(x) \leftarrow \text{false},\ p_{ue}(x) \leftarrow \text{true});[(\forall t \in T)(\neg p_{work}(t))]\{R^E\};$$

$$(\tilde{\exists}!q \in \{r,w\})p_{tip}(q))((\tilde{\forall} t \in T)p_T(t)(f_{oper}(t) \leftarrow q);$$

$$(\tilde{\forall} t \in T)p_T(t)(p_{work}(t) \leftarrow \text{true});\ [\neg((\exists t \in T)p_{work}(t))]\{R^E\};$$

$$(\tilde{\forall} t \in T)p_T(t)(p_{ue}(x) \leftarrow \text{false},\ p_{got}(x) \leftarrow \text{true},\ p_{av}(t) \leftarrow \text{true},$$

$$f_{oper}(t) \leftarrow \text{undef},\ p_{ate}(x,t) \leftarrow \text{false}); \qquad (25)$$

$$m_B = ((p_w(x) \leftarrow \text{false},\ p_{wsi}(x) \leftarrow \text{true},[(\tilde{\exists}!i \in I)p_{irand}(i)](f_{num}(x) \leftarrow i \vee R^E);$$

$$[\underline{(\tilde{\exists}!t \in T)(p_{av}(t)\ \&\ (f_{num}(x) = f_{ind}(t))}\ \&\ \underline{((\tilde{\exists}!q \in \{r,w\})p_{tip}(q))}\ \&$$

$$\neg((\exists z \in A)p_{we}(z))]\{R^E\};(f_{ati}(x) \leftarrow t,\ p_{av}(t) \leftarrow \mathit{false},$$

$$p_{wsi}(x) \leftarrow \mathit{false},\ f_{oper}(t) \leftarrow q;\ p_{usi}(x) \leftarrow \mathit{true});$$

$$[\neg p_{work}(t)]\{R^E\};\ p_{work}(t) \leftarrow \text{true};[\neg p_{work}(t)]\{R^E\};$$

$$(p_{usi}(x) \leftarrow \text{false},\ p_{got}(x) \leftarrow \text{true},\ p_{av}(t) \leftarrow \text{true},$$

$$f_{oper}(t) \leftarrow \text{undef},\ f_{ati}(x) \leftarrow \text{undef})); \qquad (26)$$

$$m_C = ((p_w(x) \leftarrow \text{false},\quad p_{ws}(x) \leftarrow \text{true});$$

$$[\underline{((\tilde{\exists}!t \in T)p_{av}(t))}\ \&\ \underline{((\tilde{\exists}!q \in \{r,w\})p_{tip}(q))}\ \&\ \neg(\exists z \in A)p_{we}(z)\ \&$$

$$\&\neg(\exists w \in A)p_{wsi}(w)]\{R^E\};(f_{ta}(t) \leftarrow x,\ p_{av}(t) \leftarrow \text{false},$$

$$p_{ws}(x) \leftarrow \mathit{false},\ f_{oper}(t) \leftarrow q,\ p_{us}(x) \leftarrow \mathit{true}\};$$

$$[\neg p_{work}(t)]\{R^E\};\ p_{work}(t) \leftarrow \text{true};[\neg p_{work}(t)]\{R^E\};$$

$$(p_{us}(x) \leftarrow \text{false},\ p_{got}(x) \leftarrow \text{true},\ p_{aw}(t) \leftarrow \text{true},$$

$$f_{oper}(t) \leftarrow \text{undef},\ f_{ta}(t) \leftarrow \text{undef})). \qquad (27)$$

В выражениях (24)–(27) используются функции и предикаты, определенные ранее для выражений (1)–(13). Для выполнения модуля m_{esis} необходимы n_A=|A| агентов-серверов $\{a_{esis}^{(1)}, a_{esis}^{(2)}, ..., a_{esis}^{(n)}\}$, образующих одно семейство.

Во фрагменте m_A записи p_{av}^{+} и p_{av}^{-} означают блокирование и разблокирование предиката p_{av} в процессе выполнения заданного фрагмента модуля. При выполнении α-итерации $[(\forall t \in T) p_{av}^{+}(t)]\{R^E\}$ сначала проверяется истинность условия в квадратных скобках. Если оно ложно, то предикат $p_{av}(t)$ разблокируется, а текущий агент-сервер снова становится в очередь за правом выбора предиката $p_{av}(t)$. Если же условие $(\forall t \in T) p_{av}(t)$ истинно, то происходит выход агента-сервера из цикла ожидания, и он выполняет все последующие действия, блокируя предикат $p_{av}(t)$ до тех пор, пока не встретится символ p_{av}^{-}.

Агенты-серверы (в совокупности составляющие «мультиагент»-сервер), интерпретируя выражения (24)–(27), конкурируют друг с другом из-за используемых ресурсов – функций и предикатов, составляющих FS-пространство. На отдельных интервалах времени агенты-серверы работают параллельно (при этом реализуется естественный параллелизм).

Заключение

Решение поставленной задачи показывает, что сети абстрактных машин обладают больш*и*ми выразительными способностями в представлении распределенных асинхронных систем, в том числе способностями представления локального управления, параллельных, конфликтных и асинхронных событий. Возможность иерархического моделирования на их основе и неинтерпретированность сетевой модели предоставляет возможность описания системы на различных уровнях абстракции и системной иерархии. Данные сети обладают способностями к реконфигурации и самомодификации, что существенно расширяет их функциональные возможности. Используемый формализм соответствует такой тенденции создания сетевого программного обеспечения, в котором стираются грани между сетевой операционной системой, распределенной системой управления базой данных и распределенным приложением, а проект «видим» разработчику на всех его этапах. Предлагаемые модели и методы могут быть использованы в качестве основы для создания новой объектно-ориентированной сетевой технологии проектирования распределенных систем хранения и обработки данных на основе согласованных взаимодействий объектов через общее пространство – коммуникационную среду или общее пространство информационных объектов.

Предложенные методы пригодны для формального описания распределенных приложений, создаваемых на базе известных технологий построения распределенных систем, а также систем управления сетями: Remote Method Invocation, Jini, JavaSpaces, Java Management Extensions, Common Object Request Broker Architecture, JXTA [14] и других технологий, в том числе технологий агентно-ориентированного программирования.

Список литературы

1. **Genrich, H. J.** Predicate/transition nets / H. J. Genrich // Lecture Notes in Computer Science. Springer-Verlag. – 1986. – V. 254. – P. 207–247.
2. **Genrich, H. J.** Equivalence transformations of PrT-Nets // Lecture Notes in Computer Science. – Springer-Verlag, 1990. – V. 424. – P. 179–208.
3. **Glaesser, U.** Combining abstract state machines with predicate/transition nets / U. Glaesser // Lecture Notes in Computer Science. Springer-Verlag. – 1997. – V. 1333. – P. 108–122.

4. **Зинкин, С. А.** Сети абстрактных машин высших порядков в проектировании систем и сетей хранения и обработки данных (базовый формализм и его расширения) // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2007. – № 3. – С. 13–22.
5. **Зинкин, С. А.** Сети абстрактных машин высших порядков в проектировании систем и сетей хранения и обработки данных (механизмы интерпретации и варианты использования) / С. А. Зинкин // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2007. – № 4. – С. 37–51.
6. **Зинкин, С. А.** Реализация барьерной синхронизации и управление процессами в виртуальном сетевом дисковом массиве / С. А. Зинкин // Информационные технологии. – 2008. – № 12. – С. 22–29.
7. **Зинкин, С. А.** Элементы новой объектно-ориентированной технологии для моделирования и реализации систем и сетей хранения и обработки данных / С. А. Зинкин // Информационные технологии. – 2008. – № 10. – С. 20–27.
8. **Глушков, В. М.** Алгебра. Языки. Программирование / В. М. Глушков, Г. Е. Цейтлин, Е. Л. Ющенко. – Киев : Наукова думка, 1978. – 320 с.
9. **Глушков, В. М.** Методы символьной мультиобработки / В. М. Глушков, Г. Е. Цейтлин, Е. Л. Ющенко. – Киев : Наукова думка, 1980. – 252 с.
10. **Капитонова, Ю. В.** Математическая теория проектирования вычислительных систем / Ю. В. Капитонова, А. А. Летичевский. – М. : Наука, 1988. – 296 с.
11. **Лавров, С. С.** Программирование. Математические основы, средства, теория. – СПб. : БХВ-Петербург, 2001. – 320 с.
12. **Волчихин, В. И.** Абстрактное и структурное моделирование сетей хранения и обработки данных / В. И. Волчихин, С. А. Зинкин // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2011. – № 4. – С. 3–18.
13. **Волчихин, В. И.** Логико-алгебраические модели и методы в проектировании функциональной архитектуры распределенных систем хранения и обработки данных / В. И. Волчихин, С. А. Зинкин // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2012. – № 2. – С. 3–16.
14. **Дейтел, Х. М.** Технологии программирования на Java 2. Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел, С. И. Сантри – М. : Бином-Пресс, 2003. – 464 с.

Зинкин Сергей Александрович
доктор технических наук, профессор,
кафедра вычислительной техники,
Пензенский государственный университет

Zinkin Sergey Alexandrovich
Doctor of engineering sciences, professor,
sub-department of computer engineering,
Penza State University

E-mail: zsa49@yandex.ru

УДК 681.324

Зинкин, С. А.

Организация управления сетями хранения и обработки данных на основе непосредственной интерпретации логико-алгебраических спецификаций / С. А. Зинкин // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2012. – № 4 (24). – С. 3–17.