DOI: https://doi.org/10.30898/1684-1719.2026.6.1

УДК: 004.33

МЕТОД СИНТЕЗА МУЛЬТИАГЕНТНОЙ АРХИТЕКТУРЫ БЕСПИЛОТНЫХ АВИАЦИОННЫХ СИСТЕМ НА ОСНОВЕ МОДУЛЬНОЙ ПРОЦЕДУРНОЙ ПАМЯТИ

И.А. Слепова, А.Л. Федер

Военно-космическая академия имени А.Ф. Можайского,
197198, Санкт-Петербург, ул. Ждановская, 13

Статья поступила в редакцию 3 мая 2026 г.

Аннотация. В статье представлен метод синтеза мультиагентной архитектуры беспилотных авиационных систем на основе модульной процедурной памяти. Целью исследования является разработка формализованного механизма динамической компоновки поведенческих модулей агентов роя беспилотных авиационных систем без применения нейросетевых методов обучения. Предложена алгебраическая модель мультиагентной системы. Каждый модуль процедурной памяти формализован как четверка, включающая предусловия, постусловия, исполнительный блок и тип синхронизации. Синтез архитектуры через эвристический алгоритм целенаправленного выбора совместимых процедур в графе зависимостей. Проведено имитационное моделирование на сценарии роя из 50 агентов. Научная новизна заключается в применении процедурной абстракции как основы динамического синтеза архитектуры роевых систем управления. Полученные результаты применимы в задачах распределенного мониторинга, поисково-спасательных операций и автономного патрулирования.

Ключевые слова: мультиагентные системы, беспилотные авиационные системы, процедурная память, роевые алгоритмы, распределенное планирование, синтез архитектуры, процедурная абстракция, динамическая компоновка, граф зависимостей, управление роем.

Автор для переписки: Федер Александр Львович, article_feder@mail.ru

Введение

Стремительное развитие беспилотных авиационных систем (БАС) в последнее десятилетие обусловило необходимость перехода от централизованных методов управления к распределенным мультиагентным архитектурам [1]. Роевые системы, включающие десятки и сотни автономных систем, демонстрируют принципиально иные требования к программному обеспечению: высокую отказоустойчивость, масштабируемость, способность к самоорганизации в условиях неполной информации о среде [2].

Существующие подходы к управлению роем БАС можно разделить на три основных класса. Первый класс – реактивные поведенческие алгоритмы типа Boids [3], основанные на локальных правилах взаимодействия агентов: разделении, выравнивании и сближении. Несмотря на вычислительную простоту, данный класс не обеспечивает выполнения сложных структурированных миссий. Второй класс – методы обучения с подкреплением (DQN, PPO, MADDPG), позволяющие синтезировать оптимальные стратегии поведения, однако требующие значительных вычислительных ресурсов для обучения и плохо поддающиеся интерпретации [4]. Третий класс – методы планирования на основе логических правил и онтологий, обладающие высокой прозрачностью, но недостаточной гибкостью при изменении сценариев [5].

Ни один из перечисленных подходов не обеспечивает одновременно:

– динамическую адаптацию поведения агентов в реальном времени;

– повторное использование поведенческих компонентов без дообучения;

– детерминированную верифицируемость логики принятия решений.

Указанные недостатки формируют актуальную проблему в науке и практике, а именно, – отсутствие на сегодняшний день метода обеспечивающего синтез мультиагентных архитектур, сочетающих модульность, динамичность и вычислительную эффективность.

Целью настоящего исследования является разработка метода синтеза мультиагентной архитектуры БАС на основе модульной процедурной памяти, обеспечивающего динамическую компоновку поведенческих модулей в режиме реального времени. Для достижения поставленной цели решаются проведены следующие этапы:

– формализация мультиагентной системы управления роем БАС;

– определение структуры модуля процедурной памяти;

– синтез алгоритма поиска и композиции процедур;

– проверка предложенного метода в условиях имитационного моделирования.

1. Формализация мультиагентной системы управления

Мультиагентная система управления роем БАС формализуется как кортеж:

$$MAS = \langle A, S, M, P, Proc \rangle, \quad (1)$$

где:

$A = \{a_1, a_2, ..., a_n\}$ – конечное множество автономных агентов (беспилотных аппаратов);

$S = S\_local \times S\_global$ – пространство состояний системы, включающее локальное состояние каждого агента (координаты, заряд аккумулятора, статус задачи) и глобальное состояние среды;

M – множество сообщений, передаваемых агентами по каналу связи в рамках топологии взаимодействия;

$P \subseteq A \times A$ – бинарное отношение, задающее топологию связности агентов;

Proc – структурированное хранилище модулей процедурной памяти, доступное всем агентам системы.

Каждый агент $a_i \in A$ характеризуется текущим состоянием $s_i \in S\_local$ и текущим набором активных процедур $A_i \subseteq Proc$. Поведение агента определяется функцией перехода:

$$\delta: S\_local \times S\_global \times M \rightarrow A_i \times Act_i, \quad (2)$$

где:

Act_i – множество доступных действий агента a_i (движение, передача данных, сбор информации и др.);

δ – функция реализует выбор процедур из Proc на основе текущего контекста.

Пространство состояний S_local каждого агента определяется вектором:

$$s_i = (x_i, y_i, z_i, v_i, \psi_i, b_i, t_i), \quad (3)$$

где:

(x_i, y_i, z_i) – пространственные координаты;

v_i – скорость;

ψ_i – курсовой угол;

$b_i \in [0,1]$ – уровень заряда аккумулятора;

t_i – идентификатор текущей задачи.

Топология связности P задается в виде графа G = (A, E), а ребра $E \subseteq A \times A$ соответствуют парам агентов, находящихся в пределах радиуса связи r_comm. Топология обновляется динамически при изменении взаимного расположения агентов, обеспечивая адаптивную маршрутизацию сообщений [6].

Глобальное состояние среды S_global формируется путем агрегации локальных наблюдений агентов и включает карту препятствий, зоны покрытия, карту обнаруженных объектов. Агрегация выполняется протоколом согласования с весовыми коэффициентами, пропорциональными достоверности локальных измерений [7]. Архитектура системы, отражающая взаимосвязь уровней, приведена на рис. 1.

Структура обеспечивает строгое разграничение компонентов архитектуры: пространства состояний, топологии связности и хранилища процедур, а введенная функция перехода δ позволяет описать механизм активации

процедур в зависимости от контекста, что является необходимым основанием для последующего синтеза метода.

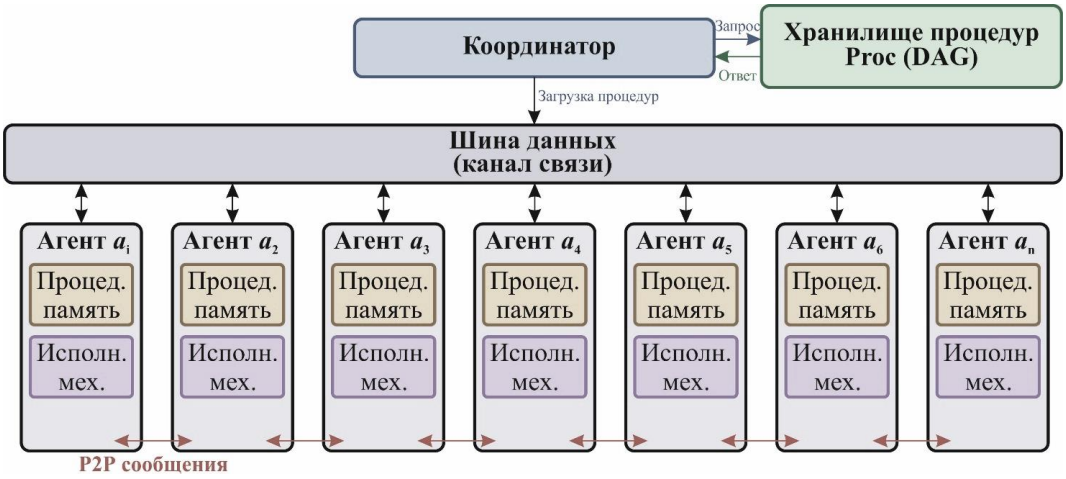

Рис. 1. Структура мультиагентной архитектуры БАС с модулями процедурной памяти.

Рассмотрим структуру метода синтеза мультиагентной архитектуры БАС на основе модульной процедурной памяти, обеспечивающего динамическую компоновку поведенческих модулей в режиме реального времени.

2. Модуль процедурной памяти

Центральным понятием предлагаемого метода является модуль процедурной памяти (МПП), структура которого представлена на рис. 2.

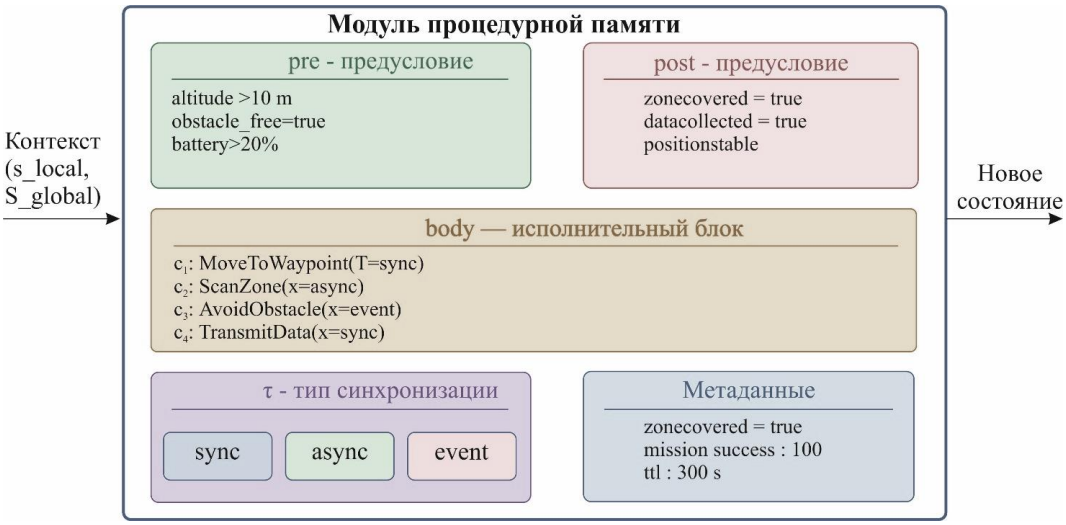

Рис. 2. Структура модуля процедурной памяти.

Модуль определяется как четверка:

$$\mathrm{MP} = (\mathrm{pre}, \mathrm{post}, \mathrm{body}, \tau), \quad (4)$$

где:

pre – предусловие активации модуля, задаваемое конъюнкцией предикатов над пространством состояний S;

post – постусловие, описывающее целевое состояние системы после выполнения модуля;

body – исполнительный блок, представляющий собой последовательность примитивных действий или вызовов других модулей (рекурсивная структура);

$\tau \in \{\mathrm{async}, \mathrm{sync}, \mathrm{event}\}$ – тип синхронизации, определяющий способ координации с другими модулями.

Предусловие pre формализуется как булева функция:

$$\mathrm{pre}: \mathrm{S\_local} \times \mathrm{S\_global} \to \{\mathrm{true}, \mathrm{false}\}. \quad (5)$$

Модуль может быть активирован агентом a_i тогда и только тогда, когда pre $(S_i, \mathrm{S\_global}) = \mathrm{true}$. Постусловие post аналогичным образом описывает состояние, которое должно быть достигнуто после выполнения body. Разрыв между post текущего модуля и pre следующего модуля является основой механизма формирования цепочек процедур.

Исполнительный блок body задается как список (возможно упорядоченный) вызовов:

$$\mathrm{body} = [c_1, c_2, \ldots, c_k],$$

где $c_j \in \mathrm{Prim} \cup \mathrm{Proc}$ – либо примитивное действие из множества Prim (маневр, отправка сообщения, захват данных), либо рекурсивный вызов другого модуля процедурной памяти. Данная структура обеспечивает иерархическую декомпозицию поведения: высокоуровневые процедуры (например, «обследование зоны») могут включать низкоуровневые (например, «облет периметра», «возврат на базу»).

Тип синхронизации τ принимает одно из трех значений. При $\tau = \mathrm{sync}$ модуль выполняется последовательно, блокируя следующие вызовы до завершения. При $\tau = \mathrm{async}$ модуль запускается параллельно

и не блокирует основной поток исполнения. При $\tau = \text{event}$ модуль активируется по внешнему событию (например, сигнал об обнаружении препятствия) [8].

3. Хранилище процедур

Хранилище процедур Proc организовано как ориентированный ациклический граф (DAG) зависимостей:

$$\text{G\_proc} = (\text{Proc}, \text{E\_dep}), \quad (6)$$

где ребра $\text{E\_dep} \subseteq \text{Proc} \times \text{Proc}$ задают отношение «предшествования»: $(\text{MP}_i, \text{MP}_j) \in \text{E\_dep}$ тогда и только тогда, когда $\text{post}(\text{MP}_i) \cap \text{pre}(\text{MP}_j) \neq \emptyset$, то есть постусловие модуля MP_i пересекается с предусловием модуля MP_j. Данная структура позволяет эффективно выполнять поиск совместимых цепочек процедур.

Оператор синтеза $\oplus$ определяется следующим образом. Для двух модулей MP_i и MP_j, удовлетворяющих условию совместимости $\text{post}(\text{MP}_i) \supseteq \text{pre}(\text{MP}_j)$, их композиция:

$$\text{MP}_i \oplus \text{MP}_j = (\text{pre}_i, \text{post}_j, \text{body}_i + + \text{body}_j, \tau\text{\_compose}(\tau_i, \tau_j)),$$

где $++$ обозначает конкатенацию исполнительных блоков, а функция $\tau\text{\_compose}: \{\text{async}, \text{sync}, \text{event}\}^2 \to \{\text{async}, \text{sync}, \text{event}\}$ задает правило выбора результирующего типа синхронизации согласно таблице приоритетов: $\text{event} > \text{sync} > \text{async}$. Операция $\oplus$ ассоциативна и не является коммутативной, что обусловлено упорядоченностью выполнения body.

Для конечной последовательности модулей $(\text{MP}_1, \text{MP}_2, \ldots, \text{MP}_m)$, удовлетворяющей цепочке совместимостей, определяется агрегированная процедура:

$\Pi = \text{MP}_1 \oplus \text{MP}_2 \oplus \ldots \oplus \text{MP}_m = (\text{pre}_1, \text{post}_m, \text{body}_1 + + \ldots + + \text{body}_m, \tau\text{\_eff})$,

где $\tau\text{\_eff}$ – результирующий тип синхронизации, вычисляемый последовательным применением $\tau\text{\_compose}$. Агрегированная процедура Π образует полный план выполнения задачи агентом [9]. Граф зависимостей G_proc с примерами процедур и ребер совместимости показан на рис. 3.

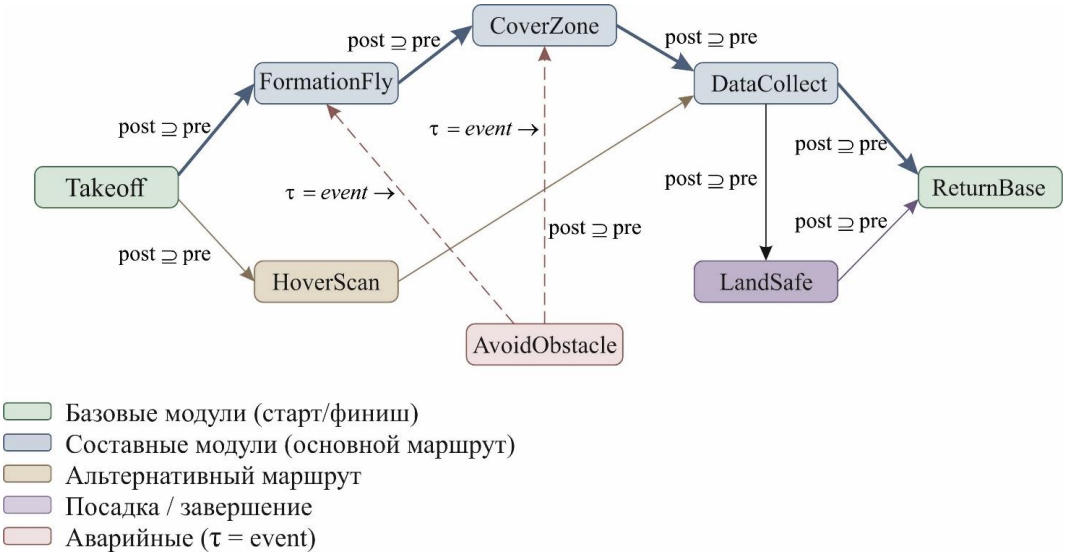

Рис. 3. Граф зависимостей G_proc модулей процедурной памяти.

4. Алгоритм поиска и компоновки процедур

Алгоритм поиска и компоновки процедур выполняется агентом a_i при поступлении новой задачи t_i от координатора и реализует эвристический целенаправленный поиск по графу G_proc. Задача t_i формализуется как целевой предикат φ_goal ⊆ S_local × S_global. Блок-схема алгоритма с таблицей τ_compose приведена на рис. 4.

Алгоритм состоит из следующих шагов:

Шаг 1 (Инициализация). Формируется начальный контекст $ctx_0 = (s_i, S\_global)$. Задается стек незакрытых целей G = {φ_goal}. Результирующий план П = ∅.

Шаг 2 (Поиск кандидатов). Для текущего контекста ctx выполняется запрос к G_proc: C = {MP ∈ Proc | pre(MP) ⊆ ctx}. Если C = ∅ – план не найден, агент запрашивает помощь у соседнего агента через шину сообщений M.

Шаг 3 (Целенаправленный выбор). Из C выбирается модуль MP* с максимальной близостью постусловия к φ_goal:

$$MP^* = \text{argmax}_{\{MP \in C\}} |post(MP) \cap \varphi\_goal| / |\varphi\_goal|.$$

Данная эвристика обеспечивает ориентацию поиска на сокращение расстояния до целевого состояния, направляя компоновку плана в сторону наиболее «полезного» в данном контексте процедурного модуля.

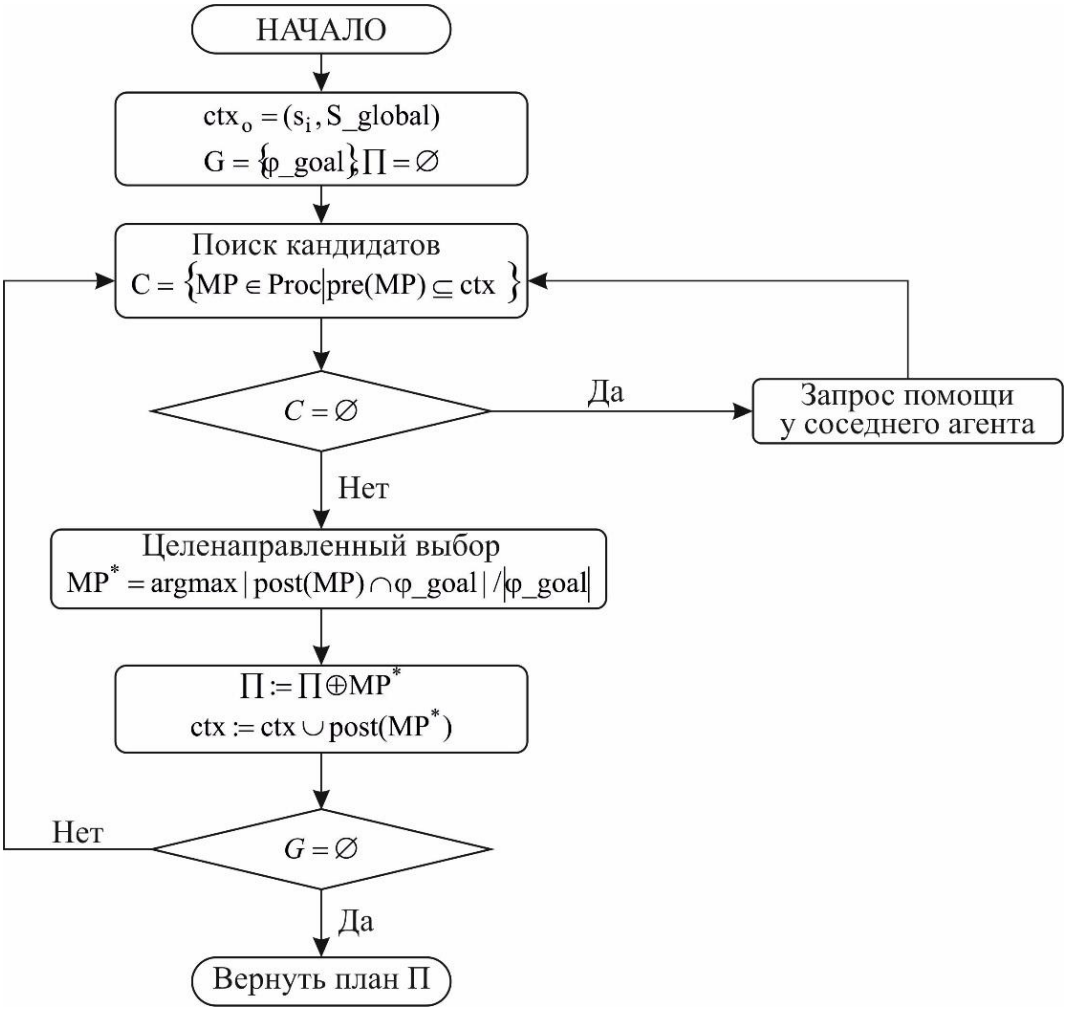

Рис. 4. Алгоритм поиска и компоновки процедур.

Шаг 4 (Обновление). Выполняется операция Π: = Π ⊕ MP*. Контекст обновляется: ctx: = ctx ∪ post(MP*). Из G удаляются предикаты, покрытые post(MP*).

Шаг 5 (Проверка завершения). Если G = ∅, возвращается план Π. Иначе возврат к шагу 2.

Вычислительная сложность алгоритма составляет O(|Proc| × |φ_goal|) в худшем случае при глубине поиска d, что значительно ниже полного перебора вариантов O(|Proc|^d). Алгоритм не гарантирует глобальную оптимальность плана, однако обеспечивает полноту в случае, когда G_proc является связным графом [10].

Механизм запроса помощи у соседнего агента (шаг 2) реализует принцип распределенного планирования: агент, не нашедший подходящей процедуры в локальном кэше, транслирует запрос ближайшим агентам, обладающим

релевантными модулями в своих кэшах. Данный механизм повышает отказоустойчивость системы и позволяет эффективно использовать специализированные процедуры, не распределяемые всем агентам [11].

Процедурная память реализует принцип разделения ответственности: агент не хранит полный перечень процедур, а поддерживает локальный кэш из k наиболее часто используемых модулей. Стратегия вытеснения кэша основана на алгоритме LFU (Least Frequently Used) с коррекцией на контекстуальную релевантность [12].

5. Метод синтеза мультиагентной архитектуры

Предлагаемый метод объединяет формальные компоненты, введенные ранее, в единую концептуальную схему метода синтеза мультиагентной архитектуры БАС на основе модульной процедурной памяти. Метод охватывает полный цикл – от постановки задачи координатором до распределенного исполнения агрегированного плана роем агентов. Метод функционирует в пяти взаимосвязанных уровнях (рис. 5).

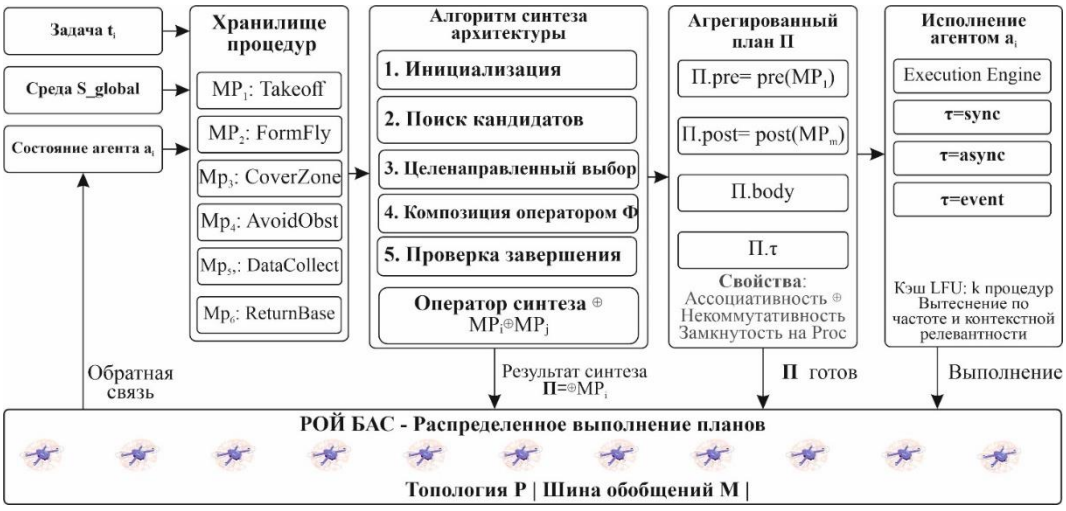

Рис. 5. Метод синтеза мультиагентной архитектуры БАС на основе модульной процедурной памяти.

Первый уровень – входные данные – формирует начальный контекст: целевой предикат φ_goal от координатора, вектор состояния агента $s_i = (x_i, y_i, z_i, v_i, \psi_i, b_i, t_i)$, и глобальное состояние среды S_global (карта препятствий, зоны покрытия).

Второй уровень – хранилище процедур Proc, организованное в виде DAG зависимостей G_proc = (Proc, E_dep), где ребра кодируют отношения совместимости модулей по условию $post(MP_i) \cap pre(MP_j) \neq \emptyset$.

Третий и центральный уровень – алгоритм синтеза – осуществляет динамическую компоновку плана П посредством пяти шагов: инициализации контекста, поиска кандидатов $C \subseteq Proc$, целенаправленного выбора оптимального модуля MP*, применения оператора синтеза $\oplus$ и проверки завершения $G = \emptyset$. Оператор синтеза формирует агрегированную процедуру согласно выражению:

$$MP_i \oplus MP_j = (pre_i, post_j, body_i + + body_j, \tau\_compose(\tau_i, \tau_j)),$$

где τ_compose подчиняется правилу приоритета event > sync > async.

Четвертый уровень – агрегированный план $П = MP_1 \oplus MP_2 \oplus ... \oplus MP_m$ – обладает свойствами ассоциативности, некоммутативности и замкнутости на множестве совместимых модулей. Вычислительная сложность построения плана составляет $O(|Proc| \times |\varphi\_goal|)$.

Пятый уровень – исполнение планов агентами роя – реализуется через Execution Engine каждого агента с применением LFU-кэша процедур.

Ключевым структурным элементом метода является блок алгоритма синтеза, в котором реализована обратная петля: при отсутствии кандидатов на шаге 2 агент инициирует распределенный запрос к соседним агентам, после получения ответа возобновляя поиск. Данная петля обеспечивает полноту алгоритма при условии связности графа G_proc. Обратная связь от роя к координатору (нижняя левая стрелка) замыкает цикл управления: результаты выполнения миссии обновляют глобальное состояние S_global и могут инициировать постановку новых задач [10, 11].

Метод допускает неоднородный состав роя: агенты с различными локальными кэшами процедур взаимодополняют друг друга через механизм распределенного запроса, не требуя централизованной синхронизации всего хранилища Proc. Это принципиально отличает предлагаемый подход от методов

на основе разделяемой глобальной памяти [13] и делает архитектуру отказоустойчивой при частичной потере связности топологии P.

6. Экспериментальные исследования

Сценарий моделирования: рой из n = {10, 20, 50, 100} агентов-квадрокоптеров выполняет миссию по покрытию прямоугольной зоны площадью 1 км² с одновременным обходом 15 случайно расположенных статических препятствий и 5 динамических объектов.

Критерии оценки:

- задержка – время от получения задачи до начала исполнения первой процедуры, (мс);
- покрытие зоны – доля покрытой площади за 300 секунд моделирования, (%);
- успешность миссии – отношение успешно выполненных миссий к общему числу запусков (100 независимых испытаний), (%);
- число столкновений за одну миссию.

Сравниваемые методы:

- предложенный метод МПП;
- реактивный алгоритм Boids с расширенным набором правил [3];
- метод глубокого обучения с подкреплением DQN (Deep Q-Network), [4,];
- метод распределенного планирования на основе HTN (Hierarchical Task Networks) [13,].

График зависимости задержки отклика от числа агентов для всех четырех методов представлен на рис. 6.

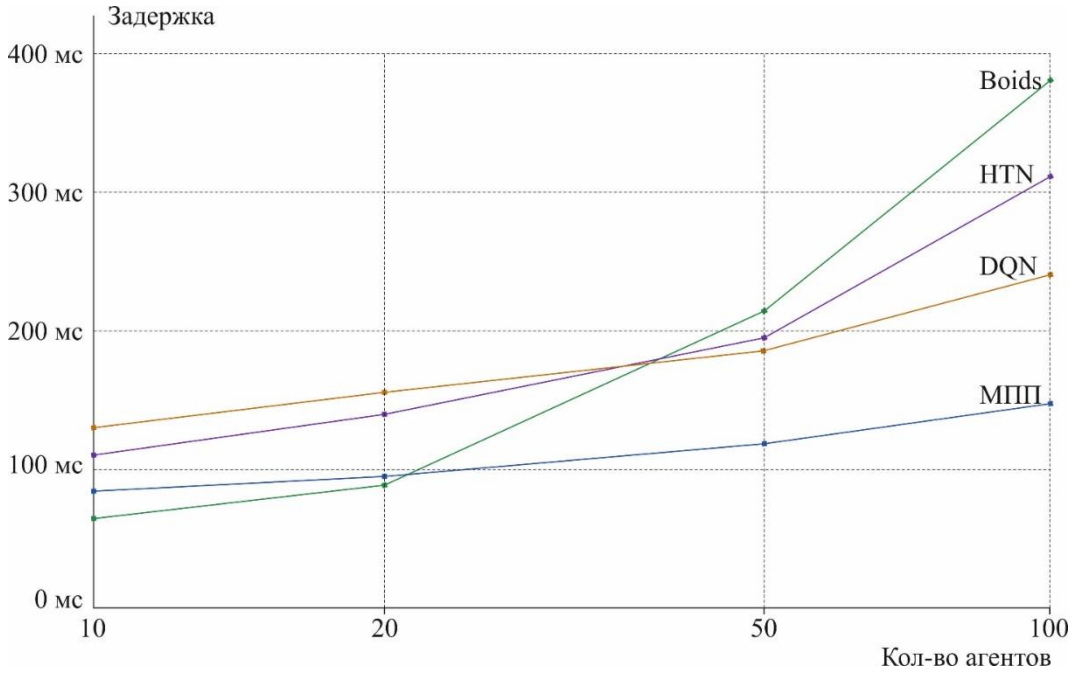

Рис. 6. Зависимость задержки отклика от числа агентов.

При числе агентов n = 50 предложенный метод МПП показал задержку отклика 120 мс, что на 35,1 % ниже, чем у метода DQN (185 мс), и на 42,9 % ниже, чем у алгоритма Boids (210 мс). Метод HTN занял промежуточное положение с задержкой 195 мс, что объясняется необходимостью полного разворота HTN-дерева при каждом изменении задачи.

Успешность выполнения миссии при n = 50 составила: МПП – 94,2 %; DQN – 89,7 %; HTN – 87,3 %; Boids – 71,4 %. Высокая успешность метода МПП объясняется применением процедур-событий для аварийного обхода препятствий, активирующихся с нулевой задержкой при обнаружении угрозы.

Покрытие зоны за 300 секунд составило при n = 50: МПП – 96,8 %; DQN – 91,2 %; HTN – 88,5 %; Boids – 82,1 %. Превосходство МПП обусловлено применением процедуры «скоординированный полет нескольких агентов в заранее определенном строю или конфигурации» («FormationFly») с динамически адаптируемым шагом сетки покрытия.

Число столкновений на миссию при n = 50: МПП – 0,12; DQN – 0,43; HTN – 0,67; Boids – 1,85. Показатель МПП в 3,6 раза ниже по сравнению с DQN благодаря явному предусловию «преодоление препятствий ≥ 2 m»

(«obstacle_clearance ≥ 2 m») в процедуре «избегайте препятствия» («AvoidObstacle»).

При масштабировании роя до n = 100 агентов метод Boids показал неприемлемый рост задержки до 380 мс вследствие квадратичной сложности вычисления локальных правил взаимодействия. Метод DQN при n = 100 показал задержку 240 мс, что объясняется ростом размерности пространства наблюдений при большом числе агентов. Метод МПП при n = 100 обеспечил задержку 148 мс, демонстрируя линейную масштабируемость O(n).

Вычислительная сложность одного цикла планирования метода МПП на процессоре Intel Core i9 при n = 50 составила в среднем 0,8 мс, что на 38,4 % ниже, чем у метода DQN (1,3 мс) и на 61,2 % ниже, чем у HTN (2,06 мс). Данный результат критически важен для встраиваемых вычислительных платформ с ограниченной производительностью. Сводное сравнение методов по пяти нормированным критериям представлено на рис. 7.

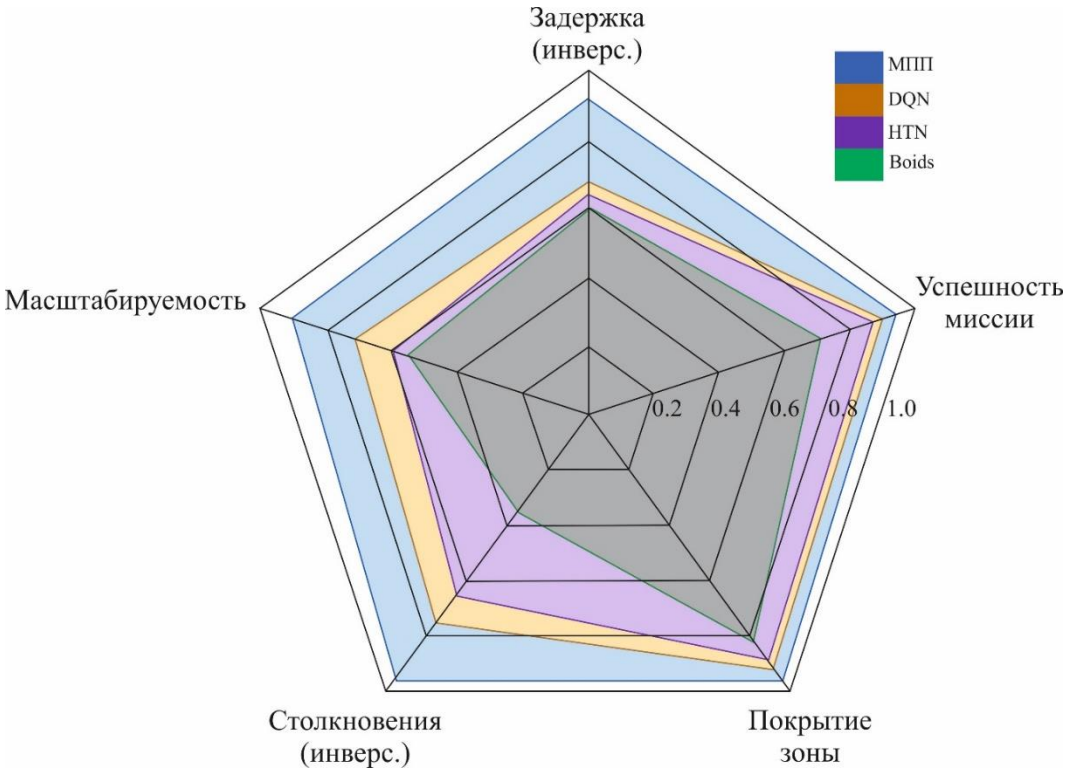

Рис. 7. Сводная диаграмма сравнения методов.

Заключение

В настоящей статье представлен метод синтеза мультиагентной архитектуры беспилотных авиационных систем на основе модульной процедурной памяти. Основными результатами являются:

1) Формальная модель мультиагентной системы в виде кортежа ⟨A, S, M, P, Proc⟩, в которой Proc представляет собой структурированное хранилище переиспользуемых процедурных модулей, организованных в DAG зависимостей. Модель обеспечивает разделение задач координации и исполнения, что повышает модульность архитектуры.

2) Модуль процедурной памяти MP = (pre, post, body, τ) и оператора синтеза $\oplus$ для динамической компоновки цепочек модулей. Оператор $\oplus$ ассоциативен, не коммутативен и замкнут на множестве совместимых модулей.

3) Алгоритм целенаправленного эвристического поиска и компоновки процедур, обеспечивающий планирование в реальном времени при наличии связного графа G_proc.

4) В ходе имитационного моделирования для роя из 50 агентов установлено, что предложенный метод обеспечивает задержку 120 мс (на 35 % ниже чем у метода DQN), успешность миссии 94,2 %, покрытие зоны 96,8 %, число столкновений 0,12 на миссию при вычислительной сложности на 38 % ниже, чем у метода DQN.

Перспективными направлениями дальнейших исследований являются: разработка онлайн-алгоритма пополнения хранилища Proc на основе накопленного опыта агентов; верификация метода на физических прототипах квадрокоптеров; интеграция МПП с методами обучения с подкреплением для автоматического синтеза предусловий и постусловий новых процедурных модулей; расширение модели на гетерогенные рои, включающие разнотипные беспилотные платформы.

Литература

1. Dorri A. et al. Multi-agent systems: A survey // Ieee Access. – 2018. – Т. 6. – С. 28573-28593. https://doi.org/10.1109/ACCESS.2018.2831228.
2. Леонов А.В., Чаплышкин В.А. Роевой интеллект для управления БПЛА в FANET // Молодой ученый. – 2016. – №. 12. – С. 314-317.
3. Цвийович А., Быковцев Ю., Манько С. Гибридный алгоритм исследования неизвестной среды мобильными роботами на основе алгоритмов ABC, PSO и Boids с механизмами памяти и межстайного обмена // Системная инженерия и инфокоммуникации. – 2026. – №. 2. – С. 11-18.
4. Тихонов М.К. Сравнительный анализ алгоритмов глубокого обучения с подкреплением DDPG, PPO и sac для управления беспилотным автомобилем в симуляторе CARLA // Научный результат. Информационные технологии. – 2024. – Т. 9. – №. 2. – С. 69-74. https://doi.org/10.18413/2518-1092-2024-9-2-0-8.
5. Скобелев П.О. и др. Применение онтологии в интеллектуальной системе распределенного управления группировкой малоразмерных космических аппаратов // Известия Самарского научного центра Российской академии наук. – 2015. – Т. 17. – №. 2-5. – С. 1119-1130.
6. Демидов Р.А., Зегжда П.Д. Унифицированная модель многоуровневых угроз нарушения информационной безопасности в сетях с динамической топологией // Интеллектуальные технологии на транспорте. – 2019. – №. 2 (18). – С. 10-14.
7. Ермилов А.С., Салтыкова О.А. Мультиагентные системы управления группой беспилотных летательных аппаратов // Наукоемкие технологии в космических исследованиях Земли. – 2025. – Т. 17. – №. 1. – С. 4-10. https://doi.org/10.36724/2409-5419-2025-17-1-4-10.
8. Van Dyke Parunak H., Brueckner S. Entropy and self-organization in multi-agent systems // Proceedings of the fifth international conference on Autonomous agents. – 2001. – С. 124-130.
9. Wooldridge M. An introduction to multiagent systems. – John wiley & sons, 2009.

10. Захаров Н.С., Харисов А.Р. Численное моделирование алгоритмов роевого управления для автономных групп беспилотных летательных аппаратов // Вестник науки. – 2025. – Т. 3. – №. 6 (87). – С. 1935-1945.
11. Петренко В.И. и др. Распределение задач в кластеризированном поле целей для гомогенных и гетерогенных групп БПЛА // Робототехника и техническая кибернетика. – 2023. – Т. 11. – №. 2. – С. 99-109. https://doi.org/10.31776/RTCJ.11203.
12. Bellemare M.G., Dabney W., Munos R.A distributional perspective on reinforcement learning // International conference on machine learning. – Pmlr, 2017. – С. 449-458.
13. Alonso-Mora J. et al. Collision avoidance for aerial vehicles in multi-agent scenarios // Autonomous Robots. – 2015. – Т. 39. – №. 1. – С. 101-121.

Для цитирования:

Слепова И.А., Федер А.Л. Метод синтеза мультиагентной архитектуры беспилотных авиационных систем на основе модульной процедурной памяти // Журнал радиоэлектроники. – 2026. – №. 6. https://doi.org/10.30898/1684-1719.2026.6.1