УДК 004.75:631.15
https://doi.org/10.31854/2307-1303-2025-13-3-1-17
EDN: NKQDLT

Сетевой модуль платформы RW.Ring

Медведев С. А.

Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича, Санкт-Петербург, 193232, Российская Федерация

Предмет и цель работы. *Технологический стек .NET Framework широко используется в сельскохозяйственных исследованиях при математическом моделировании агроэкосистем. Одним из актуальных направлений исследований являются ансамблевые расчеты. Для проведения таких исследований требуется легковесный механизм удаленного вызова процедур, обеспечивающий эффективное сетевое взаимодействие приложений.* ***Методы****: технологический стек .NET Framework с ранее разработанными библиотеками платформы RW.Ring; объектно-ориентированное программирование; приемы работы с сетевыми протоколами TCP и HTTP; методы построения сервис-ориентированной архитектуры.* ***Результат.*** *Разработанный модуль поддерживает разные способы сетевого взаимодействия: механизм удаленного вызова процедур и бинарных команд, а также их интеграцию в клиент-серверную архитектуру. Подчеркнуты преимущества модуля по сравнению с технологией WCF: высокая производительность, компактность кода, минимизация вычислительных ресурсов и гибкость настройки под различные задачи. Модуль поддерживает протоколы TCP и HTTP, что позволяет разработчикам адаптировать его для обработки больших объемов данных, включая сериализацию и аутентификацию пользователей. Рассмотрены примеры применения модуля в исследованиях агроэкосистем, где он обеспечивает анализ влияния климатических изменений на урожайность культур, минимизируя затраты на полевые эксперименты. Модуль упрощает интеграцию в распределенные системы, оптимизирует использование ресурсов и поддерживает оперативное сопровождение полевых исследований.* ***Научная новизна*** *заключается в создании легковесной альтернативы традиционным технологиям сетевого взаимодействия, что снижает издержки и повышает производительность. Предложен более универсальный механизм бинарных команд, дополняющий традиционную сервис-ориентированную архитектуру.* ***Практическая значимость*** *проявляется в повышении эффективности работы исследователей, работающих с большими данными, и в поддержке принятия решений в сельском хозяйстве. Модуль позволяет интегрироваться с системами дистанционного зондирования для автоматического анализа фотоснимков, что расширяет его применение в агропромышленном комплексе. Платформа RW.Ring вносит значительный вклад в развитие цифровых технологий для сельского хозяйства, предлагая инновационный подход к организации распределенных вычислений и сетевого взаимодействия, что делает ее перспективным инструментом для международных научных коллабораций.*

Ключевые слова: *машинное обучение, теория массового обслуживания, нагрузка, математическое моделирование, миграция, балансировка*

Библиографическая ссылка на статью:
Медведев С. А. Сетевой модуль платформы RW.Ring // Информационные технологии и телекоммуникации. 2025. Т. 13. № 3. С. 1–17. DOI: 10.31854/2307-1303-2025-13-3-1-17. EDN: NKQDLT

Reference for citation:
Medvedev S. Network Module of the RW.Ring Platform // Telecom IT. 2025. Vol. 13. Iss. 3. PP. 1–17. (in Russian). DOI: 10.31854/2307-1303-2025-13-3-1-17. EDN: NKQDLT

Введение

Модели продукционного процесса сельскохозяйственных культур позволяют решать исследовательские задачи в области агропромышленного комплекса, существенно снизив затраты на проведение полевых опытов [1]. При использовании регрессионных моделей можно получать результаты, укладывающиеся в тренды, наблюдаемые в ходе многолетних полевых исследований [2]. Применение динамических механистических моделей продукционного процесса позволяет выйти за рамки этих трендов и получать результаты для условий, которые изначально не встречались в ходе полевых исследований. Именно поэтому динамические механистические модели продукционного процесса сельскохозяйственных культур используются в исследованиях влияния климата будущего на поведение агроэкосистем [3]. Однако, как правило, вычислительная сложность таких экспериментов столь высока, что даже сниженная трудоемкость, по сравнению с полевыми опытами, может быть значительной. Здесь речь идет не столько о количестве труда исследователя, сколько о машинном времени, требуемом для выполнения масштабных модельных расчетов. В связи с этим, проводить исследования климата будущего с использованием динамических моделей продукционного процесса на персональных компьютерах нецелесообразно. Гораздо эффективнее выделить для этого специализированные мощные машины и подключаться к ним извне.

Кроме того, такая клиент-серверная архитектура позволяет повысить эффективность работы исследователя, который работает не только с моделями продукционного процесса, но и с живыми посевами, так как информационное обеспечение модели требует разворачивания большого количества программных компонентов, в то время как небольшое клиентское приложение, запускающее поливариантный расчет динамических моделей продукционного процесса, может быть установлено относительно легко. При необходимости можно даже сделать мобильное приложение, которое позволит исследователю проводить оперативное сопровождение полевого опыта или производственного посева, находясь непосредственно в поле или в камералке [4]. Также сервисы могут быть использованы для повышения эффективности дистанционного зондирования. Можно предположить, что современные работы, посвященные автоматическому анализу фотоснимков, могут быть использованы совместно с сетевыми технологиями, чтобы обеспечить интеллектуальный автоматизированный ввод данных дистанционного зондирования в системы поддержки принятия решений в сельском хозяйстве [5].

Помимо повышения эффективности работы индивидуального исследователя, клиент-серверная архитектура приложений активно используется для коллаборации ученых. Так, проект ICASA [6] содержит в себе множество Web-сервисов, обеспечивающих возможность проведения различных исследований с применением динамических моделей продукционного процесса ученым со всего мира. Одни и те же сервисы, выставленные в Интернет, могут быть использованы широким кругом ученых из разных научных групп [7]. Поэтому и плат-

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

форма RW.Ring [8] содержит в себе решение типовых задач, связанных с сетевым взаимодействием приложений.

В принципе, технологический стек .NET, на которой сделана платформа RW.Ring, содержит в своем составе технологию WCF, которая, в общем и целом, предназначена для решения именно тех задач, с которыми исследователь сталкивается при разработке сетевых взаимодействий [9]. Однако, несмотря на наличие готового решения этой задачи, на практике это решение показалось авторам данной работы неудовлетворительным, и было принято решение о создании собственного сетевого модуля. На ранних этапах работы он представлял собой ряд надстроек над WCF, которые позволили бы сделать работу с этой технологией более комфортной, но в итоге было принято решение о полном отказе от WCF. Дело в то, что WCF делает очень много лишнего и этим создает множество неочевидных проблем, всплывающих лишь задним числом. Например, если на сервере бросить исключение безопасности, невозможно сделать так, чтобы текст сообщения об ошибке дошел до клиента, WCF любое сообщение об ошибке подменяет текстом «Отказано в доступе». Кроме того, эта технология основана на не слишком эффективной концепции сообщений, которая приводит к тому, что элементарное сетевое взаимодействие требует значительного количества ресурсов. Технология, обеспечивающая высокоуровневые сетевые взаимодействия, должна быть не инфраструктурой, стремящейся полностью абстрагировать код приложения от низкоуровневых API, а набором надстроек над ними для решения различных прикладных задач. Дело в том, что какой бы сложной ни была технология, которая пытается все абстрагировать, в ряде задач может возникнуть ситуация, когда для реализации недостающей возможности все равно необходимо опуститься на более низкий уровень. В частности, вместо сообщений гораздо эффективнее использовать потоки данных, так как это позволяет работать с транспортными потоками напрямую с минимальным перерасходом ресурсов [10].

Материалы и методы

Платформа RW.Ring содержит сетевой модуль, основанный на более базовых API .NET Framework, находящихся в библиотеке System.dll и представляющий собой обертку над базовыми сетевыми функциями Windows. В отличие от WCF, которая поддерживает все, что может понадобиться широкому кругу разработчиков, и в результате – переусложнена, сетевой модуль в RW.Ring имеет минимальный необходимый функционал, который допускается при необходимости расширить. Следует отметить, что эти библиотеки входят в самый минимальный набор стандартных библиотек .NET Framework Client Profile, что снижает системные требования платформы RW.Ring до минимума.

При всех недостатках технологии WCF, заменить которую призван сетевой модуль RW.Ring, в ней есть ряд правильных и эффективных решений. Соответствующие функции в сетевом модуле были сделаны аналогично. Примером такого решения является описание контрактов служб и данных с помощью специальных атрибутов. Еще одним эффективным решением является меха-

низм прозрачных прокси, встроенных в самые базовые библиотеки технологического стека .NET. Этот механизм позволяет работать с удаленным объектом, вызывая его методы на клиенте так же, как будто этот объект является локальным объектом. Правда, этот механизм используется не только в WCF, но и в более ранней технологии .NET Remoting, на смену которой пришла WCF [11]. Прозрачный прокси при вызове его методов обращается к связанному с ним экземпляру реального прокси, передавая ему описание каждого вызова в виде специальной объектной модели. Реальный прокси может быть реализован разработчиком таким образом, чтобы из этой объектной модели вызова сформировать обращение к какому-нибудь внешнему API, например сокетам для связи с удаленным сервером. После этого реальный прокси должен вернуть в виде специальной объектной модели результат выполненной операции, которую прозрачный прокси преобразует в возвращаемое значение и выходные параметры вызванного метода. Таким образом, прозрачный прокси обеспечивает возможность прямого вызова методов в коде, а реальный прокси позволяет разработчику делать ту реализацию этих вызовов, которая ему требуется. Если подобный механизм используется для вызова одноименных методов удаленного объекта на сервере, такой подход называются удаленным вызовом процедур (RPC, *аббр. от англ.* Remote Procedure Call) [10].

На стороне сервера работает не прокси, а хост. Это объект, который прослушивает входящие клиентские подключения и обрабатывает их. В случае удаленного вызова процедуры хост должен определить, что за метод вызван с клиента, и какому объекту он принадлежит, а затем обратиться к соответствующему объекту на сервере, вызвав его метод с параметрами, полученными от клиента. Результаты выполнения метода хост должен отправить клиенту обратно. Для отправки данных по сети от клиента к серверу и от сервера к клиенту необходимо информацию о вызываемом методе и его параметрах, а также о результате вызова метода, преобразовывать в такой формат, который пригоден для передачи по сети, с использованием сериализации. Таким образом, любое клиент-серверное взаимодействие можно описать схемой, представленной на рисунке 1.

Схема клиент-серверного взаимодействия предполагает:

1) формирование клиентского запроса;
2) сериализацию клиентского запроса;
3) передачу сериализованных данных по сети;
4) десериализацию клиентского запроса на сервере;
5) выполнение клиентского запроса на сервере;
6) формирование ответа от сервера;
7) сериализация ответа от сервера;
8) передачу сериализованных данных по сети;
9) десериализацию ответа от сервера на клиенте;
10) обработку ответа от сервера.

Из приведенной схемы (см. рисунок 1) видно, что клиентское приложение отправляет запрос и получает определенный ответ в виде объектной модели. Серверное приложение получает запрос и отправляет ответ в виде той же самой объектной модели. Именно это взаимное соответствие запросов и отве-

тов на клиенте и на сервере обеспечивает возможность удаленного вызова процедур. Ниже уровнем лежит сериализатор, который преобразует объектную модель запроса и ответа в представление данных, которое может быть использовано для передачи по сети (например, JSON), и обратно. Сериализованные данные передаются с использованием сокетов – объектов операционной системы, обеспечивающих взаимодействие с сетевыми устройствами. Очевидно, что для согласованной работы клиент-серверных приложений необходимо использование одного и того же сериализатора как на сервере, так и на клиенте.

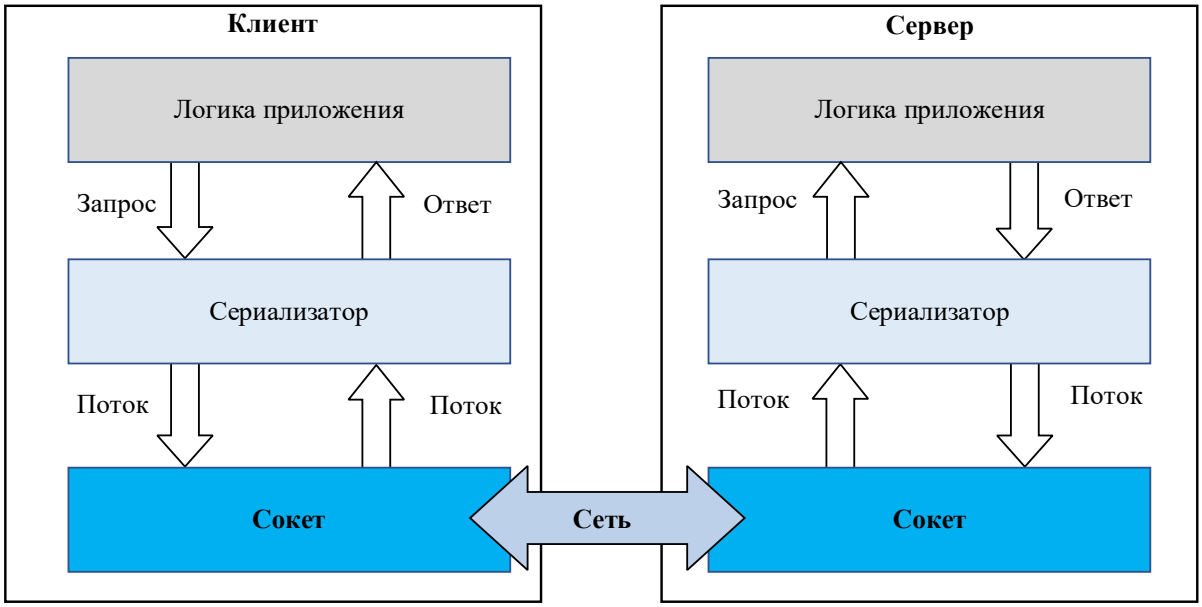

Рис. 1. Схема клиент-серверного взаимодействия

Однако, в отличие от WCF, сетевой модуль RW.Ring, помимо удаленного вызова процедур, имеет второй механизм – бинарные команды. Он используется тогда, когда для обмена данными между клиентом и сервером не требуется сериализация, поскольку запрос и ответ и так уже представляют собой бинарные данные, готовые к отправке по сети. Вне зависимости от необходимости выделенной сериализации, для описанного взаимодействия достаточно потокового транспортного протокола. Тем не менее, использование более высокоуровневых механизмов, позволяющих работать с прикладным протоколом HTTP, позволяет самостоятельно не решать ряд задач, связанных с распределением нагрузки, так как предоставляет готовое решение, расположенное либо в системных объектах Windows (на сервере), либо в стандартной библиотеке .NET Framework (на клиенте). Дело в том, что потоковые транспортные протоколы имеют достаточно ресурсозатратные операции открытия и закрытия подключения, особенно при установлении защищенного соединения, и необходимо избегать того, чтобы при выполнении приложением частых запросов подключение открывалось и закрывалось каждый раз при вызове удаленного метода. Поэтому для более эффективной работы необходимо организовать пул подключений, который будет позволять использовать одно открытое подключение многократно и закрывать его лишь по тайм-ауту. Это приводит к определенным

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

сложностям при сериализации, требующих специальной буферизации потоков входных и выходных данных. При использовании компонентов для работы с протоколом HTTP стандартная библиотека .NET предоставляет готовое решение всех этих задач. Еще одним преимуществом от использования прикладного протокола HTTP является возможность применения описанных в этом протоколе абстракций для более полного отражения семантики запросов. Поэтому в сетевом модуле RW.Ring поддерживается два протокола: максимально простой и легкий механизм на базе транспортного протокола TCP и более сложный – на базе прикладного протокола HTTP. Произвольно комбинируя протоколы и прикладные механизмы, можно получить четыре возможных режима работы сетевого модуля RW.Ring: удаленный вызов процедуры TCP и HTTP, а также бинарные команды TCP и HTTP.

Результаты и обсуждение

В основе удаленного вызова процедур лежит объектная модель вызовов методов, которая предполагает возможность сериализации для передачи по сети. Как и в WCF, для сериализации этой информации используются контракты данных. Аналогом контракта службы является интерфейс, помеченный специальным атрибутом `RpcService`. Опционально у этого атрибута через параметр конструктора может быть задано имя для идентификации сервиса; если оно не задано, используется имя интерфейса. Методы этого интерфейса, предназначенные для удаленного вызова, должны быть помечены атрибутом `RpcOperation`. У этого атрибута так же есть опционально задаваемое через параметр конструктора имя, которое используется вместо реального имени метода интерфейса, если это требуется. Дело в том, что имена всех методов в пределах одного контракта службы должны быть уникальными, и, если в интерфейсе есть перегруженные методы, их логическое переименование через атрибут может разрешить конфликт. Имя метода может содержать латинские буквы в верхнем или нижнем регистре, знаки подчеркивания и цифры, и не может начинаться с цифры.

Обработка этой информации происходит в классах `RpcServiceInfo`, один из которых является обобщенным, а второй нет. Основная логика находится в обобщенном классе, который реализует интерфейс `IRpcServiceInfo`.

Интерфейс `IRpcServiceInfo` описывает следующие элементы:

– `ServiceName` (логическое имя сервиса, либо совпадающее с именем интерфейса, либо указанное в атрибуте RpcService);

– `GetMethodName` (получение логического имени метода, либо совпадающего с реальным именем метода, переданного в качестве параметра, либо указанное в атрибуте `RpcOperation`);

– `GetOperationInfo` (получение специального объекта, описывающего операцию по имени метода; этот объект имеет тип `RpcOperationInfo`);

– `HasMethod` (проверка наличия метода по логическому имени).

Класс `RpcOperationInfo` представляет собой метод интерфейса контракта сервиса (`Method`), логическое свойство, показывающее наличие у метода вы-

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

ходных или ссылочных параметров (HasReferenceParameters), а также типы, используемый для сериализации запроса к серверу (RequestType), ответа от сервера (ResponseType) и для передачи результатов операции (ResultType).

Поскольку метод может иметь множество параметров, а сериализаторы всегда работают с одним объектом, для агрегации нескольких параметров в один объект используется специальный класс ParametersList, используемый для вычисления свойства RequestType. Этот класс имеет два статических метода: Create и GetRequiredType.

По методу Create и значениям его параметров в виде массива объектов генерируется экземпляр интерфейса IParametersList, представляющий собой один объект, в который агрегированы все эти параметры, предназначенный для эффективной сериализации. Метод GetRequiredType предназначен для получения конкретного типа данных, реализующего интерфейс IParametersList, экземпляр которого возвращает метод Create.

В свою очередь, интерфейс IParameterList имеет два метода: возвращающий массив типов параметров метода, для которого создан экземпляр интерфейса (GetTypes), и – значения параметров, передаваемых методу, в виде массива объектов (GetValues).

Свойство RpcOperationInfo.ResultType вычисляется, исходя из типа возвращаемого значения метода и наличия у этого метода выходных или ссылочных параметров. Если таких параметров нет, то тип, используемый для передачи результата, совпадает с типом возвращаемого значения. В противном случае используется обобщенный тип ReturnWithOut, содержащий тип возвращаемого значения метода и типы его параметров, так как при удалённом вызове необходимо получать от сервера и то, и другое. Если у метода нет возвращаемого значения, но есть выходные или ссылочные параметры, то для передачи результата используется тот же тип, что и для сериализации запроса.

Свойство RpcOperationInfo.ResponseType всегда возвращает обобщенный тип CallResult, которому в качестве единственного параметра передан тот тип, который используется для передачи результатов операции. Помимо значения этого типа, CallResult содержит сведения об ошибке, которая могла возникнуть при выполнении операции. Класс реализует необобщенный интерфейс, представляющий собой объект, содержащий результат операции (Value) – тип RpcOperationInfo.ResultType, а также сведения об ошибке (Error), имеющие специальный тип исключения (ClientServerException), позволяющий узнать на клиенте серверный стек вызовов, которые привели к ошибке.

Обобщенная версия класса RpcServiceInfo реализует паттерн Singleton и имеет статическое свойство Instance, которое возвращает экземпляр этого класса для текущего типа сервиса. Благодаря этому весь анализ интерфейса осуществляется при инициализации, а дальше работа идет с уже подготовленными данными, обеспечивающими наилучшую производительность. Необобщенная версия класса RpcServiceInfo является статической и представляет собой обертку над обобщенной для слаботипизированных сценариев. В частно-

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

сти, она позволяет получать описание сервиса по его логическому имени, что необходимо для обработки клиентских запросов хостами.

Очевидно, что основная задача описанной подсистемы сетевого модуля RW.Ring состоит в том, чтобы подготовить на клиенте и на сервере согласованные наборы данных для сериализации. Сама сериализация осуществляется через интерфейсы `ISerializer` и `ISerializationFactory`. Интерфейс `ISerializer` представляет собой сериализатор для конкретного типа данных – индивидуальный для каждого запроса к серверу и ответа от него. В нем объявлены методы `Serialize` и `Deserialize`, сигнатура которых совпадает с сигнатурой соответствующих методов стандартных сериализаторов. Метод `Serialize` пишет сериализованные данные из потока байт, метод `Deserialize` читает данные из потока байт. Интерфейс `ISerializationFactory` содержит метод `GetSerializer`, принимающий в качестве параметра тип данных, для которого необходимо создать сериализатор, и возвращающий экземпляр `ISerializer` для этого типа данных.

Связь между бинарными командами и удаленным вызовом процедур на клиенте осуществляется через интерфейс `IClientCaller`. Именно разные реализации этого интерфейса позволяют переключаться между сетевыми протоколами (TCP или HTTP). Первый метод этого интерфейса `Call` в качестве параметров принимает логическое имя метода контракта службы, значения параметров этого метода в виде `IParametersList` и описание этой операции в виде `RpcOperationInfo`; он возвращает экземпляр интерфейса `ICallResult`, содержащий либо результат удаленного вызова метода, либо информацию об ошибке при его вызове. Два других метода интерфейса `IClientCaller` представляют собой варианты обмена бинарными данными между клиентом и сервером: через потоки и через массивы байт. Метод `BinaryExchange` в качестве параметров принимает текстовую строку с командой, которую требуется выполнить, и массив байт – данные для этой команды. Он возвращает массив байт с результатами этой команды. Для больших объемов данных предназначен метод `StreamExchange`, который делает то же самое, но с потоками данных. Во избежание полного дублирования больших потоков данных, он не возвращает значения и имеет три параметра: 1) выполняемую команду; 2) делегат, пишущий данные в поток запроса; 3) делегат, читающий данные из потока ответа. Именно это решение позволяет максимально эффективно использовать потоковые транспортные протоколы для передачи больших объемов данных по сети. В WCF для этого использовались контракты операций с потоками, но сделать это в рамках архитектуры удаленного вызова процедур эффективно невозможно. Именно поэтому в сетевом модуле RW.Ring эта задача решается посредством отдельного API. На сервере эти запросы будут обрабатываться одноименными методами `IBinaryService`. Метод `BinaryExchange` имеет такую же сигнатуру, как и на клиенте, а метод `StreamExchange` принимает в качестве параметров текст выполняемой команды и два бинарных потока, один – для чтения данных, пришедших от клиента, другой – для отправки данных клиенту. На стороне сервера этому интерфейсу соответствует класс `ServiceHostBase`. Дело в том,

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

что большое количество логики работы самого хоста не зависит от протокола, а часть требует переопределения. В связи с этим базовый класс хоста содержит большое количество защищенных методов, часть из которых является абстрактными и требует переопределения в производных классах. Взаимодействие клиента с хостом показано на рисунке 2.

Общая логика работы хоста может быть описана следующими действиями.

1) Хосту при инициализации передается фабрика сериализации и конечная точка, на которой он будет прослушивать входящие соединения (конструктор класса).

2) Указывается, какие контракты служб будет обслуживать хост; для этого ему передаются классы, в которых реализованы интерфейсы этих контрактов (метод `AddService`). Также с хостом может быть связан сервис, обслуживающий бинарные команды (свойство `BinaryService`).

3) Запуск прослушивания входящих соединений на конечной точке, с которой связан хост (методы `Start` и `StartService`). Ожидание входящего соединения осуществляется в фоновом потоке.

4) Когда приходит входящее подключение, из пула потоков извлекается еще один поток для обработки входящего соединения (методы `GetState` и `ProcessRequest`).

5) В потоке для обработки входящего соединения хост читает заголовок команды и определяет, что это было за действие с клиентской стороны, соответствующее одному из трех методов интерфейса `IClientCaller`. Если это был Call, то происходит десериализация и выполнение метода, два остальных метода перенаправляются связанному с хостом сервису бинарных команд. После получения результата он либо сериализуется, либо передается в сетевые потоки напрямую, для чего используются методы `GetCaller`, `GetSerializer`, `BinaryExchange` и `StreamExchange`.

6) При завершении работы хоста необходимо остановить прослушивание конечной точки и освободить все используемые хостом ресурсы операционной системы (метод `Dispose`).

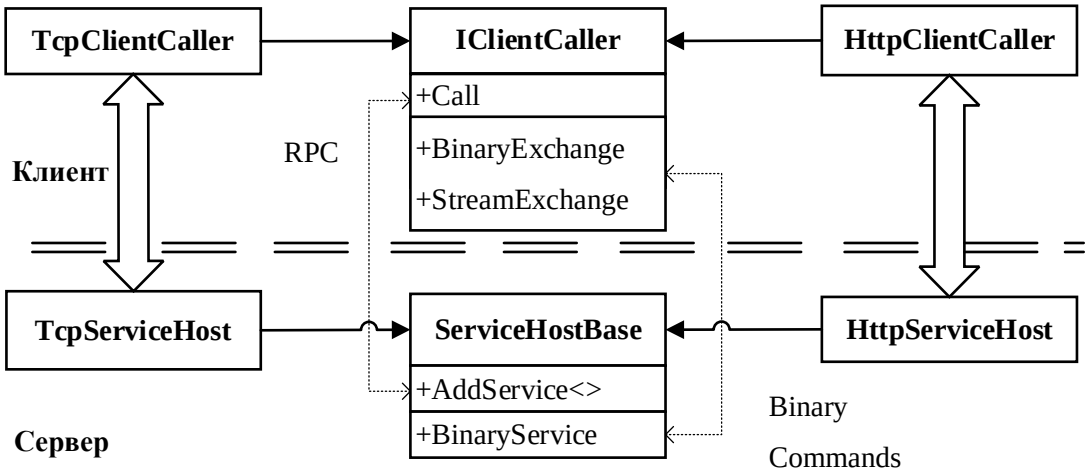

Рис. 2. Взаимодействие реализаций базовых сетевых интерфейсов

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

Для работы удаленного вызова процедур, помимо описанных подсистем, необходимы прокси. Сетевой модуль RW.Ring содержит два класса прокси. Класс `GenericProxy` наследуется от класса стандартной библиотеки `RealProxy` и представляет собой его обобщенную версию, более удобную в использовании. В частности, метод `GetTransparentProxy` в этом классе сделан обобщенным, чтобы возвращать объект сразу нужного типа, кроме того, он позволяет в экземпляре прозрачного прокси реализовывать необходимые интерфейсы. Этот класс имеет абстрактный метод `Invoke`, принимающий в качестве параметра описание вызванного метода `IMethodCallMessage` и возвращающий экземпляр класса `ReturnMessage`. Этот метод переопределен во втором классе прокси `NetworkProxy`, который использует экземпляр интерфейса `IClientCaller` для удаленного вызова методов. Используя класс `RpcServiceInfo`, он преобразует экземпляр стандартного интерфейса `IMethodCallMessage` в сериализуемые параметры метода `IClientCaller.Call`, описанные выше, после чего преобразует десериализованный экземпляр `ICallResult`, пришедший от метода `IClientCaller.Call`, в экземпляр `ReturnMessage`, который должен вернуть этот метод прокси.

На сервере работу того же самого уровня выполняет специальный класс `ServerCaller`. Для своей работы он требует фабрику экземпляров конкретного сервиса, которая может либо каждый раз возвращать один и тот же экземпляр, либо на каждое обращение создавать свой. Метод `Call` у этого класса принимает в качестве параметров описание вызванного с клиента метода `RpcOperationInfo` и значения параметров этого метода `IParametersList`. Он возвращает `ICallResult`, содержащий результаты вызова указанного метода у объекта, порожденного переданной экземпляру данного класса фабрикой. Класс `ServerCaller` используется следующим образом: для каждой службы, размещенной на хосте, создается свой экземпляр этого класса, доступный по логическому имени службы через защищенный метод.

Таким образом, при удаленном вызове процедур выполняется набор задач.

1) Преобразование описаний вызовов методов и их результатов в виде объектов типов из стандартной библиотеки в сериализуемые объекты и обратно. На клиенте эту задачу выполняет класс `NetworkProxy`, на сервере `ServerCaller`.

2) Сериализация и десериализация подготовленных для этого описаний вызовов методов и их результатов в формат, пригодный для передачи по сети. За это отвечают реализации интерфейсов `ISerializer` и `ISerializationFactory`.

3) Отправка и получение сериализованных данных через сеть. На клиенте эту задачу выполняют реализации интерфейса `IClientCaller` (`TcpClientCaller` и `HttpClientCaller`), на сервере – наследники класса `ServiceHostBase` (`TcpServiceHost` и `HttpServiceHost`).

4) Классы `NetworkProxy` и `ServiceHostBase` используют подготовленную структурированную информацию о методах сервиса, доступную через класс `RpcServiceInfo`.

Схематически взаимодействие этих подсистем показано на рисунке 3.

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

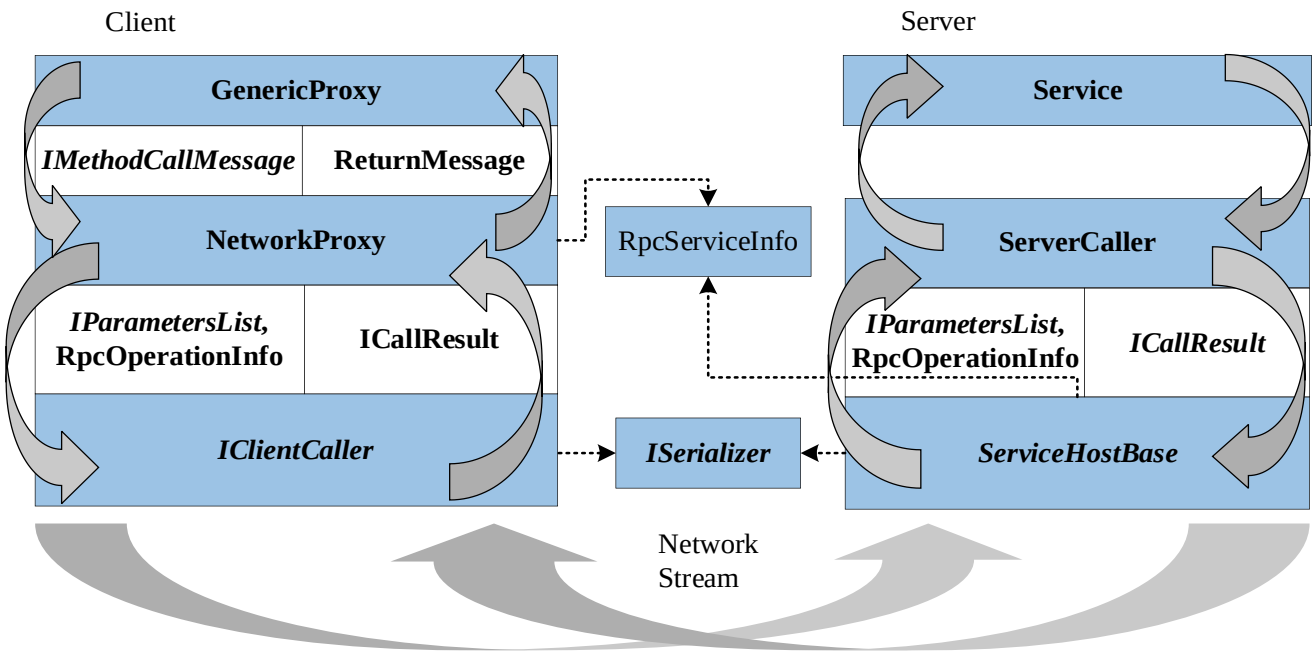

Рис. 3. Принципиальная схема работы удаленного вызова процедур в сетевом модуле RW.Ring

Механизм бинарных команд позволяет работать с сетевыми потоками напрямую. Однако поверх него сетевой модуль RW.Ring предлагает более высокоуровневый инструмент, содержащий собственный сериализатор. В основе него лежит обобщенный класс `RemotableCommand`. Команда создается на клиенте, сериализуется и отправляется на сервер, на сервере она десериализуется и выполняется, после чего клиенту отправляется результат выполнения этой команды в сериализуемом виде. Для работы этого механизма необходимо, чтобы все команды были описаны в сборках, доступных как на клиенте, так и на сервере. Параметр обобщенного класса команды – это тип результата, который должен быть унаследован от класса `RemotableResult`. Этот класс содержит следующие свойства: success – логическое свойство, показывающее, было ли выполнение команды успешным; exception – ошибка, которая могла возникнуть в ходе выполнения команды на сервере (свойство имеет тип `ClientServerException`).

Сам класс `RemotableCommand` содержит вызываемый на клиенте метод `Execute`, который не имеет параметров и возвращает результат указанного типа. На сервере будут вызываться переопределенные методы класса `CreateEmptyResult` и `Fill`. Первый из них создает экземпляр результата выполнения команды на сервере, второй выполняет основную работу и в переданный ему созданный первым методом результат выполнения команды записывает все, что требуется. Для обращения к серверу метод `Execute` использует статическое свойство `RemoteCaller.DefaultCaller`, которое предоставляет точку доступа к экземпляру интерфейса `IRemotableCaller`, через который команда может быть выполнена. Этот интерфейс имеет единственный метод `Call`, принимающий в качестве параметра команду и возвращающий результат ее выполнения. Он имеет две реализации. Первая из них, класс `RemotableCallerStub`, выполняет команду в текущем окружении без обраще-

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

ния к серверу, вызывая ее методы `CreateEmptyResult` и `Fill`. Вторая реализация этого интерфейса, `RemoteCaller`, сериализует команду сериализатором, который ему передан через параметры конструктора, и отправляет полученные данные на сервер, используя метод интерфейса IClientCaller StreamExchange. Его экземпляр также передается классу RemoteCaller через параметры конструктора.

На сервере команды будут обрабатываться методом `StreamExhange` класса `BinaryCommandService`. Этот метод предполагает, что его первый параметр с текстом команды – это класс команды, для которой нужно создать сериализатор. Тип сериализатора в виде `ISerializationFactory`, как и в случае с `RemoteCaller`, передается этому классу через параметр конструктора. Он десериализует команду из входного потока, вызывает ее методы `CreateEmptyResult` и `Fill`, и сериализует сформированный результат выполнения команды в выходной поток данных. Таким образом, механизм бинарных команд гораздо проще, чем механизм удаленного вызова процедур. Описанная схема его работы показана на рисунке 4.

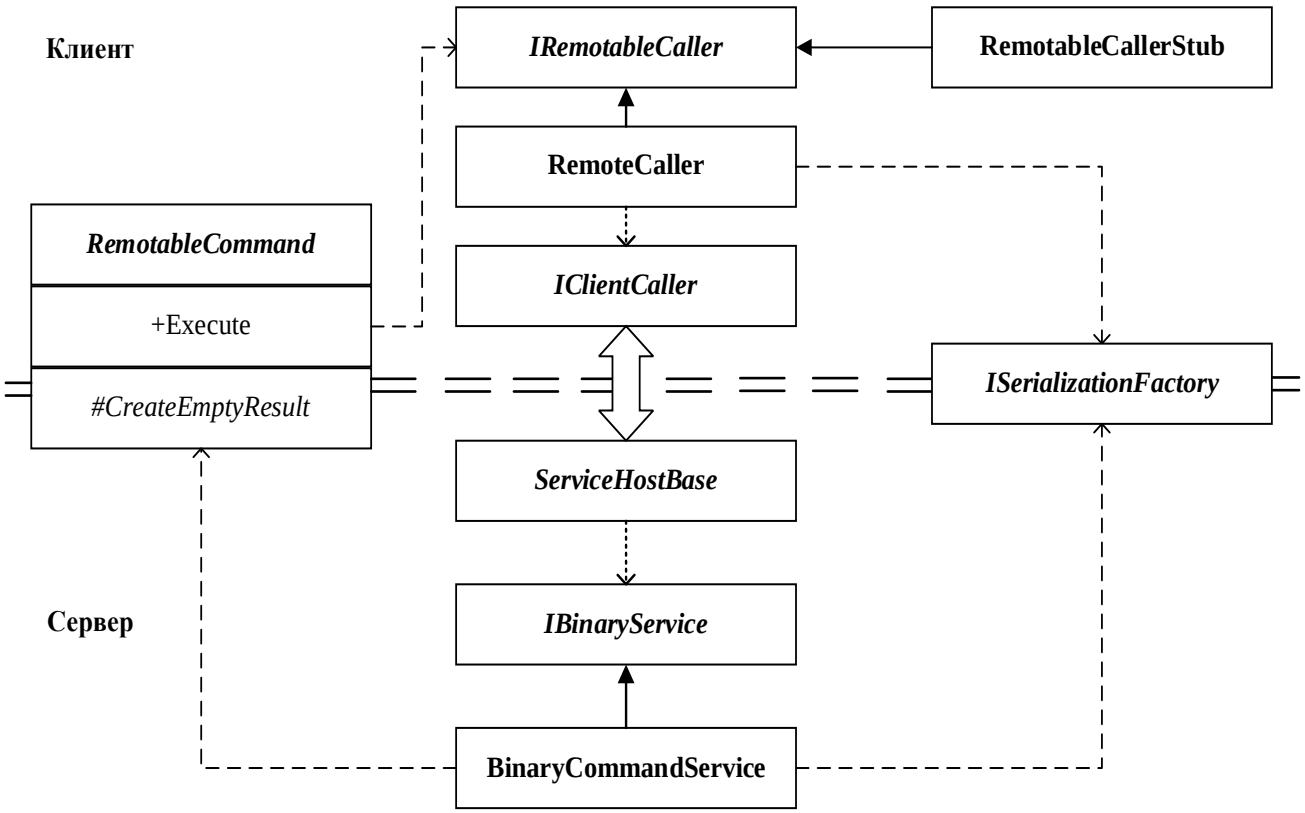

Рис. 4. Схема работы механизма бинарных команд в сетевом модуле RW.Ring

Еще одна из задач, которая должна решаться в ходе разработки клиент-серверных приложений – это аутентификация пользователя. Поскольку сетевой модуль RW.Ring поддерживает как работу напрямую по протоколу TCP, так и использование более сложного прикладного протокола HTTP, в зависимости от этого механизм передачи данных варьируется. Но в обоих случаях на уровне объектной модели используется специальный класс `ClientInfo`.

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

Классу `ClientInfo` содержит следующие свойства: `Application` – приложение, которое подключается к серверу; `UserName` – имя пользователя, запустившего приложение-клиент; `MachimeName` – имя машины, на которой запущено приложение-клиент. Кроме того, этот класс имеет два статических свойства: `ProcessInfo` и `ThreadInfo`. `ProcessInfo` всегда возвращает информацию об окружении текущего процесса, а `ThreadInfo` позволяет задать и получить информацию для текущего потока. При этом автоматически перестраиваются контексты протоколирования, чтобы в логах отображалась информация о том, что те или иные действия были инициированы определенным клиентом. При выполнении запроса клиентские реализации интерфейса `IClientCaller` присоединяют к сериализованным данным информацию о клиенте. На сервере наследники класса `ServiceHostBase` десериализуют экземпляр `ClientInfo` и помещают его в значение свойства `ThreadInfo` для текущего потока на время выполнения запроса от пользователя. При использовании протокола TCP экземпляр `ClientInfo` передается перед полезными данными в виде строки специального вида, напоминающего упрощенный JSON, а при использовании протокола HTTP информация сохраняется в HTTP-заголовках Authorization, User-Agent и Machine-name. Эта информация может быть использована фабриками экземпляров служб, чтобы создавать сеансовые службы, привязанные к конкретным клиентам.

Заключение

В настоящее время все актуальнее становится проблема «раздувания» программного обеспечения. В то время как модельные расчеты, связанные с проблемой изменения климата действительно требуют большого количества вычислительных ресурсов, многие современные технологии приводят к созданию приложений, которые требуют их не в силу высокой вычислительной сложности, а в силу недостаточной оптимизации программных продуктов. Особенно остро эта проблема стоит именно в области Интернет-программирования, поскольку современные Web-сайты зачастую так перегружены сложными библиотеками, что достаточно простой функционал все равно требует большого количества вычислительных ресурсов. Созданный сетевой модуль демонстрирует противоположный подход, при котором технология содержит не абстракцию от нижележащего уровня, а именно надстройку над ним. Предназначенный для небольшого круга задач, связанного с созданием распределенных систем ансамблевых расчетов по динамическим механистическим моделям продукционного процесса [12], он не перегружен большим объемом кода, который не используется. Легкий доступ к Web-запросам и ответам, к сериализаторам и сокетам позволяет программировать на любом из требуемых уровней, что позволит оптимизировать приложения для высоких нагрузок. Многие проекты по исследованию поведения агроэкосистем в различных условия с помощью динамических моделей, итак, требуют большого количества вычислительных ресурсов, и архитектура сетевого модуля RW.Ring сделана таким образом, чтобы как можно меньше это усугублять.

DOI: 10.31854/2307-1303-2025-13-3-1-17
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_3/1-17.pdf

Благодарим главного программиста ООО «ОДИТЕК» Сергея Николаевича Грамницкого за глубокое исследование возможностей и недостатков WCF, которое позволило прийти к осознанию необходимости данной работы.

Литература

1. Poluektov R.A., Fintushal S.M., Oparina I.V., Shatskikh D.V., Terleev V.V., et al. Agrotool – A system for crop simulation // Archives of Agronomy and Soil Science. 2002. Vol. 48. Iss. 6. PP. 609–635. DOI: 10.1080/0365034021000041597. EDN: PWGBSR

2. Antoniadou T., Wallach D. Evaluating Decision Rules for Nitrogen Fertili-Zation // Biometrics. 2000. Vol. 56. Iss. 2. PP. 420–426. DOI: 10.1111/j.0006-341X. 2000.00420.x. EDN: FOXTOT

3. Palosuo T., Hoffmann M.P., Rötter R.P., Lehtonen H.S. Sustainable intensification of crop production under alternative future changes in climate and technology: The case of the North Savo region // Agricultural Systems. 2021. Vol. 190. P. 103135. DOI: 10.1016/j.agsy.2021.103135. EDN: HXFNNC

4. Anacleto R., Figueiredo L., Almeida A., Novais P. Server to Mobile Device Communication: A Case Study // Proceedings of the 4th International Symposium on Ambient Intelligence – Software and Applications. Advances in Intelligent Systems and Computing. Heidelberg: Springer, 2013. Vol. 219. PP. 79–86. DOI: 10.1007/978-3-319-00566-9_11

5. Самойлов А.Н., Бородянский Ю.М., Волошин А.В. Метод и распределенная индуктивная процедура машинного обучения фотограмметрического алгоритма для решения задач определения геометрических параметров объектов по предварительно обработанным цифровым изображениям // Инженерный вестник Дона. 2020. № 12(72). С. 220–230. EDN: JPKSEI

6. Jones J.W., Keating B.A., Porter C.H. Approaches to modular model development // Agricultural Systems. 2001. Vol. 70. Iss. 2-3. PP. 421–443. DOI: 10.1016/S0308-521X(01)00054-3

7. Van De Glind G., Brynte C., Skutle A., Kaye S., Konstenius M., et al. The International Collaboration on ADHD and Substance Abuse (ICASA): Mission, Results, and Future Activities // European Addiction Research. 2020. Vol. 26. Iss. 4-5. PP. 173–178. DOI: 10.1159/000508870. EDN: DMWBGM

8. Medvedev S., Terleev V., Vasilyeva O. Non-visual platform components for a system of polyvariant calculation of dynamic models of the production process // Proceedings of the XXII International Scientific Conference Energy Management of Municipal Facilities and Sustainable Energy Technologies. E3S Web Conf. 2021. Vol. 244. P. 09008. DOI: 10.1051/e3sconf/202124409008. EDN: KPEVFO

9. Gastermann B., Stopper M. Windows Communication Foundation hosting methods for distributed industrial applications // Annals of DAAAM and Proceedings of the 20th International DAAAM Symposium «Intelligent Manufacturing & Automation: Focus on Theory, Practice and Education». Vienna: DAAAM Internat., 2009. PP. 1925–1926.

10. van Renesse R., Tanenbaum A.S., van Staveren H., Hall J. Connecting RPC-Based Distributed Systems using Wide-Area Networks // Proceedings of the 7th International Conference on Distributed Computing Systems. IEEE, 1987. PP. 28–34.

11. Wiener R. Remoting in C# and .NET // Journal of Object Technology. 2004. Vol. 3. Iss. 1. PP. 83–100. DOI: 10.5381/jot.2004.3.1.c8

12. Медведев С.А., Черяев А.С. Перспективы создания универсального сервиса удалённых ансамблевых расчётов динамических моделей продукционного процесса культурных растений // Агрофизика. 2020. № 3. С. 45–52. DOI: 10.25695/AGRPH.2020.03.07. EDN: FOXJMR

Статья поступила 05 октября 2025 г.
Одобрена после рецензирования 14 ноября 2025 г.
Принята к публикации 17 ноября 2025 г.

Информация об авторе

Медведев Сергей Алексеевич – кандидат сельскохозяйственных наук, доцент кафедры систем обработки данных Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича.
E-mail: medvedev1.sa@sut.ru

Network Module of the RW.Ring Platform

Medvedev S.

The Bonch-Bruevich Saint Petersburg State University of Telecommunications,
St. Petersburg, 193232, Russian Federation

Purpose. *The .NET Framework technology stack is widely used in agricultural research for mathematical modeling of agroecosystems. One of the most relevant areas of research is ensemble computations. Conducting such research requires a lightweight remote procedure call (RPC) mechanism that enables efficient network communication between applications.* ***Methods***. *.NET Framework technology stack with previously developed RW.Ring platform libraries; object-oriented programming; techniques for working with TCP and HTTP network protocols; methods for building service-oriented architecture.* ***Results***. *The developed module supports various methods of network interaction: remote procedure calls (RPC) and binary commands, as well as their integration into the client-server architecture. The module's advantages over WCF technology are highlighted: high performance, code compactness, minimization of computational resources, and flexibility in adapting to various tasks. The module supports TCP and HTTP protocols, enabling developers to tailor it for processing large data volumes, including serialization and user authentication. Examples of the module's application in agroecosystem research are provided, where it enables analysis of climate change impacts on crop yields while reducing the costs of field experiments. The module simplifies integration into distributed systems, optimizes resource usage, and supports real-time field research. The* ***scientific novelty*** *lies in developing a lightweight alternative to traditional network interaction technologies, reducing overhead and enhancing performance. A more universal binary command mechanism is proposed to complement the traditional service-oriented architecture. Its* ***practical significance*** *is evident in improving the efficiency of researchers handling large datasets and supporting decision-making in agriculture. The module integrates with remote sensing systems for automated photo analysis, expanding its applications in the agroindustrial sector. The RW.Ring platform significantly contributes to the development of digital technologies for agriculture, offering an innovative approach to organizing distributed computations and network interactions, making it a promising tool for international scientific collaborations.*

Key words: *network module, RW.Ring platform, client-server architecture, remote procedure call, binary commands, serialization, authentication, agroecosystems, modeling, performance*

References

1. Poluektov R.A., Fintushal S.M., Oparina I.V., Shatskikh D.V., Terleev V.V., et al. Agrotool – A system for crop simulation. *Archives of Agronomy and Soil Science,* 2002, vol. 48, iss. 6, pp. 609–635. DOI: 10.1080/0365034021000041597. EDN: PWGBSR

2. Antoniadou T., Wallach D. Evaluating Decision Rules for Nitrogen Fertili-Zation. *Biometrics,* 2000, vol. 56, iss. 2, pp. 420–426. DOI: 10.1111/j.0006-341X. 2000.00420.x. EDN: FOXTOT

3. Palosuo T., Hoffmann M.P., Rötter R.P., Lehtonen H.S. Sustainable intensification of crop production under alternative future changes in climate and technology: The case of the North Savo region. *Agricultural Systems,* 2021, vol. 190, p. 103135. DOI: 10.1016/j.agsy.2021.103135. EDN: HXFNNC

4. Anacleto R., Figueiredo L., Almeida A., Novais P. Server to Mobile Device Communication: A Case Study. *Proceedings of the 4th International Symposium on Ambient Intelligence – Software and Applications. Advances in Intelligent Systems and Computing, vol. 219*. Heidelberg: Springer, 2013, pp. 79–86. DOI: 10.1007/978-3-319-00566-9_11

5. Samoylov A.N., Borodyansky Y.M., Voloshin A.V. Method and distributed inductive procedure of machine learning of a photogrammetric algorithm for solving the problems of determining the geometric parameters of objects by pre-processed digital and digital images. *Engineering journal of Don*. 2020, no. 12(72), pp. 220–230. (in Russian) EDN: JPKSEI

6. Jones J.W., Keating B.A., Porter C.H. Approaches to modular model development. *Agricultural Systems*. 2001, vol. 70, iss. 2-3, pp. 421–443. DOI: 10.1016/S0308-521X(01)00054-3

7. Van De Glind G., Brynte C., Skutle A., Kaye S., Konstenius M., et al. The International Collaboration on ADHD and Substance Abuse (ICASA): Mission, Results, and Future Activities. *European Addiction Research*. 2020, vol. 26, iss. 4-5, pp. 173–178. DOI: 10.1159/000508870. EDN: DMWBGM

8. Medvedev S., Terleev V., Vasilyeva O. Non-visual platform components for a system of polyvariant calculation of dynamic models of the production process. *Proceedings of the XXII International Scientific Conference Energy Management of Municipal Facilities and Sustainable Energy Technologies. E3S Web Conf., vol. 244*. 2021, p. 09008. DOI: 10.1051/e3sconf/202124409008. EDN: KPEVFO

9. Gastermann B., Stopper M. Windows Communication Foundation hosting methods for distributed industrial applications. *Annals of DAAAM and Proceedings of the 20th International DAAAM Symposium "Intelligent Manufacturing & Automation: Focus on Theory, Practice and Education"*. Vienna: DAAAM Internat., 2009, pp. 1925–1926.

10. van Renesse R., Tanenbaum A.S., van Staveren H., Hall J. Connecting RPC-Based Distributed Systems using Wide-Area Networks. *Proceedings of the 7th International Conference on Distributed Computing Systems*. IEEE, 1987, pp. 28–34.

11. Wiener R. Remoting in C# and .NET. *Journal of Object Technology*. 2004, vol. 3, iss. 1, pp. 83–100. DOI: 10.5381/jot.2004.3.1.c8

12. Medvedev S.A., Cherayev A.S. Prospects for creating universal service for remote ensemble calculations of dynamic models of cultivated plant production process. *Agrophysica*. 2020, no. 3, pp. 45–52. (in Russian) DOI: 10.25695/AGRPH.2020.03.07. EDN: FOXJMR

Information about Authors

Medvedev Sergey – Candidate of Agricultural Sciences. Associate Professor at the Department of Data Processing (The Bonch-Bruevich Saint Petersburg State University of Telecommunications). E-mail: medvedev1.sa@sut.ru