DOI: 10.15514/ISPRAS–2020–32(6)–5

# О разработке Оберон-системы с заданными свойствами эргодичности

*Д.В. Дагаев, ORCID: 0000-0003-0343-3912 <dvdagaev@oberon.org>*
*АО «Русатом – Автоматизированные системы управления»,*
*115230, Россия, г.Москва, Каширское шоссе, д. 3, корп. 2, стр. 16*

**Аннотация.** Эргодичность систем характеризуется сохранением стационарности распределения состояний. Для компьютерных систем важно не допустить деградацию свойств системы со временем. Эргодичность особенно необходима для критически важных систем в ответственных отраслях. Разработка ПО на основе требований функциональной безопасности стандарта МЭК 60880 категории А реализуется только на вновь создаваемом ПО, отвечающим данным, наиболее жестким требованиям для АЭС, при невозможности использовать стандартные ОС и компиляторы. Для данных целей был реализован прототип среды исполнения и прикладного ПО дисплейной системы командного управления (ДСКУ). Среда исполнения (рантайм) создавалась на основе Active Oberon системы A2. A2 представляет собой однопользовательскую многозадачную систему. Область применения – промышленные встроенные системы реального времени, системы повышенной надежности. Среда исполнения ДСКУ реализована существенной переработкой минимального подмножества A2 для обеспечения требований стандарта. Система ограничений, формируемая по требованиям стандарта, дает возможность создавать компьютерные системы с новыми свойствами. Использование данных ограничений приводит к доказательству отсутствия возможности появления вызываемых ими сбоев и позволяет рассматривать компьютерную систему исходя из презумпции неэргодичности.

**Ключевые слова:** эргодичность; Оберон; надежность

**Для цитирования:** Дагаев Д.В. О разработке Оберон-системы с заданными свойствами эргодичности. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 67-78. DOI: 10.15514/ISPRAS–2020–32(6)–5

**Благодарности:** Международный общественный научно-образовательный проект Информатика-21.

# Towards Developing of Oberon System with Specific Requirements of Ergodicity

*D.V. Dagaev, ORCID: 0000-0003-0343-3912 < dvdagaev@oberon.org >*
*Rusatom – Automated Control Systems JSC,*
*Kashirskoye Shosse, 3/2/16, Moscow, Russian Federation, 115230*

**Abstract.** The ergodic system keeps the time average is the same for almost all initial points. It is important for computer systems to prevent the degradation of the properties of the system over time. Ergodicity is especially required for mission-critical systems in demanding industries. Software development based on the functional safety requirements of the IEC 60880 category A standard is implemented only on newly created software that meets the most stringent requirements for nuclear power plants, it is impossible to use standard operating systems and compilers. For these purposes, a prototype of the runtime environment and application software of the command display system (DSCU) was implemented. The runtime was created based on the Active Oberon A2 system. A2 is a single-user multi-tasking system. Application area - industrial embedded real-time systems, high reliability systems. The DSKU execution environment is implemented by a significant revision of the minimum subset A2 to meet the requirements of the standard. The system of restrictions formed according to the requirements of the standard makes it possible to create computer systems with new properties. The use of





these constraints leads to the proof that there is no possibility of the occurrence of the failures they cause and allows us to consider a computer system based on the presumption of non-ergodicity. This «via negative» approach is based on restrictions, the addition of which allows one to obtain new qualitative properties. The more restrictions, the greater the gain in system reliability and stability.

**Keywords:** ergodicity; Oberon; reliability

**For citation:** Dagaev D.V. Towards developing of Oberon system with specific requirements of ergodicity. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 6, 2020, pp. 67-78 (in Russian). DOI: 10.15514/ISPRAS–2020–32(6)–5

**Acknowledgements**. Informatika-21 (a non-commercial project to promulgate scientific rationality for IT education).

## 1. Введение

Мы постоянно сталкиваемся с разного рода примерами, когда функционирующая минуту назад компьютерная система неожиданно переходит в какой-то режим, при котором нормальная работа невозможна или, как минимум, затруднена. Вот произошел сбой, мы увидели «синий экран смерти», и в результате имеем ту же ОС, то же прикладное программное обеспечение, но в данном режиме изменилось важное – состояние. Пользователь вынужден вмешаться, нажать кнопки аппаратного рестарта и начать процесс функционирования заново. При работе с Windows часто происходят зависания, приводящие к деградации выполнения компьютером функций пользовательского интерфейса. Каждый такой случай имеет свои первопричины, но в конечном итоге пользователь определяет, что данное состояние с его точки зрения недопустимо, и перезагружает компьютер. Примеров таких множество, и особенно неприятно, когда это происходит с автономными устройствами: роутерами, ТВ-приставками, контроллерами АСУ ТП.

Независимо от первопричин такого поведения систем самое общее рассмотрение показывает, что в компьютерной системе со временем изменяются свойства, относящиеся к программному обеспечению. Это может быть используемые ресурсы, как динамическая память и процессорное время. Это могут быть ресурсы ПО, как количество открытых файлов или потоков. Изменение этих свойств определяется изменением состояния системы в целом. Плохо, если изменение состояния со временем приводит к изменению свойств в сторону ухудшения (например, меньше динамически запрашиваемой памяти).

Отсюда и вытекает рассмотрение эргодичности компьютерных систем как описания процесса изменения состояния и деградации свойств. Важны также и способы предотвращения деградации свойств.

Существуют разные системы требований, относящиеся к свойствам внедряемых компьютерных систем, для АЭС они определены в перечне отраслевых международных стандартов. Требования стандартов устанавливают конкретные ограничения, тогда как эргодичность определяет общие классифицируемые черты и описывает тенденции происходящих процессов.

В данной статье описываются проблемы деградации свойств и рассматриваются способы их решения на примере разработанной системы ДСКУ средствами Active Oberon на базе рантайма A2.

## 2. Свойство эргодичности применительно к программным системам

Когда мы говорим об эргодичности, имеется в виду стационарность, вероятности $\alpha i$ состояний системы не зависят от времени и не зависят от распределения вероятностей в начальный момент времени, то есть: $\alpha i = const$.

В работах [1, 2] показана важность свойства эргодичности, при невыполнении которого распределение состояний нестационарных систем будет существенно отличаться от

начальных состояний со временем. Авторы доказывают, что для таких систем среднее по времени и в количественном выражении, и в динамике, существенно отличаются. Это обусловлено тем, что для неэргодических систем распределения вероятностей начинают зависеть от предыстории, и случайная величина перестает быть независимой.

Свойство эргодичности применимо и к программным системам. Например, разработано серверное ПО и тестируется требования на безотказность в течении 100 часов работы. Рассматриваются два способа проверки:

- одно серверное приложение тестируется непрерывно в течении 100 часов (по времени);
- 100 приложений тестируются по 1 часу каждое в автономной среде (по ансамблю).

Если в разработанном серверном ПО есть утечки памяти, то со временем используемая память будет расти, а свойства системы (свободная память) – деградировать. В связи с этим фактором вероятность отказа со временем будет расти, система становится не эргодической, и вероятность отказа за 100 часов непрерывной работы будет больше вероятности отказа по ансамблю. С точки зрения тестировщика такой случай является абсолютно неравноценной заменой, и в требованиях обычно пишут 100 часов непрерывной работы.

В данной работе будет идти разговор об эргодичности применительно к общему состоянию системы и о конкретных свойствах (например, свободная память), поведение которых будет описываться как поведение свойств эргодичности.

В идеализированной программной системе постулируются допущения об обязательном успешном выделении памяти за минимальное время, отсутствии ограничений стека, реальном времени без блокировок и гонок и нулевой стоимости синхронизации, системе исключений, вызывающей безопасный рестарт, идеальной сети с отсутствием потерь, с нулевой латентностью и отсутствием переполнений счетчиков времени. В реальной системе обоснованность этих допущений требует доказательств, в том числе и архитектурного характера, из которых можно сделать вывод об эргодичности. Далее будут перечисляться допущения с большей степенью детализации.

Задачей разработчиков надежного ПО является получение предсказуемых свойств эргодичности. Если данные свойства находятся в допустимых рамках, это гарантирует отсутствие вызываемых ими программных сбоев. Идею профилактики вместо хирургии высказывал еще Н.И.Пирогов: «будущее принадлежит медицине предохранительной» [3].

## *3. Требования функциональной безопасности к ДСКУ*

Разработка ПО на основе требований функциональной безопасности стандарта МЭК 60880 категории А [4] реализуется только на вновь создаваемом ПО, отвечающим наиболее жестким требованиям для АЭС, при невозможности использовать стандартные ОС и компиляторы. В части ОС необходимо, как минимум, следующее:

- *B.2cb:* следует избегать использования универсального операционного программного обеспечения (операционных систем);
- *B.2cc:* если операционная система необходима, то ее применение следует ограничить небольшим числом простых функций;
- *B.2cd:* операционная система должна содержать только необходимые функции.

В части реального времени и времени обработки требуется, как минимум, следующее:

- *B.2dd:* время прогона не должно существенно изменяться в результате изменения входных данных;
- *B.2de:* значение изменения времени прогона, которое может быть вызвано входными данными, должно быть документально оформлено;
- *B.2dg:* объем данных, считываемых в течение одного вычислительного цикла, должен быть постоянным.





В части работы с памятью требуется, как минимум, следующее:

- *B.3c:* содержимое памяти должно быть защищено или контролироваться;
- *B.3db:* следует проверять правильность передачи любого параметра, включая проверку типа параметров;
- *B.3dc:* при адресации массива следует проверять его границы.

В части работы с прерываниями требуется, как минимум, следующее:

- *B.2e:* необходимо ограничивать применение прерываний;
- *B.2ea*: прерывания могут использоваться, если они упрощают проект ПО и не делают верификацию чрезмерно сложной;
- *B.2ed*: если прерывания используются, то для непрерываемых частей необходимо иметь оценки максимального времени вычислений, чтобы можно было рассчитать максимальное время, в течение которого прерывание запрещено.

Требования данного стандарта носят конкретный характер, однако они хорошо подпадают под определение свойств эргодичности.

Для данного набора требований необходимо было реализовать прототип среды исполнения и прикладного ПО дисплейной системы командного управления (ДСКУ). Прикладное ПО должно было поддерживать функции как контроллера в АСУ ТП, так и как человеко-машинного интерфейса (ЧМИ) со средствами индивидуального управления задвижками и насосами, встраиваемый в стойки промышленного исполнения АСУ ТП.

## *4. Реализация на основе Active Oberon системы A2*

### 4.1 Оберон-система A2

Научная школа Н. Вирта в лице Швейцарской высшей технической школы Цюриха (Swiss Federal Institute of Technology Zurich, ETH Zurich) и научно-образовательного проекта Информатика-21[1] развивает уже 50-летнюю историю эволюционирования языков программирования Алгол-Паскаль-Модула-Оберон; поэтому выбор для средств для реализации прототипа был не случайным.

Среда исполнения (рантайм) создавалась на основе разработанной для X86 версии Active Oberon-системы A2[2]. A2 представляет собой однопользовательскую многозадачную систему. Область применения – промышленные встроенные системы реального времени, системы повышенной надежности. Архитектуру A2 можно назвать наноядерной, т.к. A2 реализует рантайм «на голом железе» – «bare machine» [5]. При этом имеется минимальные модули рантайма – Objects, Machine, драйверы и механизмы переключения приоритетный режим – пользовательский режим.

Среда исполнения ДСКУ реализована путем существенной переработки минимального подмножества A2 для обеспечения требований стандарта. Исключены все функции, не являющиеся необходимыми. Исключены средства поддержки многоядерности и потенциально проблемные области рантайма.

### 4.2 Модель памяти A2 и управление памятью ДСКУ

При рассмотрении вопросов работы с памятью делается допущение #1 – представление о неограниченности памяти:

1) отсутствуют утечки памяти;

[1] http://www.inr.ac.ru/~info21/
[2] https://github.com/metacore/A2OS

2) динамическая память не фрагментирована;
3) запрашиваемая аллокируемая память может и будет выделена;
4) время срабатывания сборщика мусора равно нулю;
5) пределы выделения стека отсутствуют;
6) своппинг срабатывает идеально за нулевое время;
7) постраничная организация памяти не оказывает воздействия на временные характеристика системы;
8) статическая память гарантированно выделяется при рестарте компонентов системы.

Модель памяти A2 основана на стандартных механизмах сегментации. При этом динамическая виртуальная память отображается на физическую с сохранением адресов [5], исключая области нулевых адресов, стека и специальных устройств. Таким образом, при обращении к нулевому указателю или к стеку за пределами адресуемой памяти формируется ошибка сегментации. Авторы A2 называют обращение к виртуальной памяти «ложной абстракцией» [5], при которой подкачка страниц по запросу не осуществляется.

Модель памяти A2 гарантирует правильность допущений #1.6 (отсутствие своппинга) и #1.7 (отсутствие постраничной организации памяти).

В системе ДСКУ реализован запрет управления памятью после фазы инициализации. Цикл работы системы разбит на 6 этапов:

1) загрузка системы;
2) фаза инициализации: объектам и данным предоставляется динамическая память; происходит выделение запрашиваемой стековой памяти (рекурсия отсутствует);
3) запрет выделения памяти, отключение сборщика мусора, запрет дисковых операций;
4) код старта активных объектов;
5) цикл обновления по таймеру;
6) окончание работы и перезагрузка.

После этапа 3 все операции по выделению динамической памяти прекращаются. Запрет выделения памяти означает установку программного исключения при каждом возможном обращении. На уровне рантайма нормальная работа происходит на этапе 5. Восстанавливаемые прерывания приводят к коду рестарта этапа 4 с переходом на 5. Фатальные ошибки приводят к этапу 6 – перезагрузке системы.

Данный механизм доказывает справедливость допущений #1.1-#1.5, #1.8 на этапах, начиная с 4. Если фаза инициализации пройдена, корректная работа с памятью становится доказанной. Система функционирует с фиксированным объемом динамической и стековой памяти, которые были выделены на этапе инициализации.

## 4.3 Реальное время и активные объекты

Реальное время и синхронизация соответствуют допущению #2:

1) отсутствуют синхронизационные тупики (deadlock);
2) нет гонок; вызванные ими искажения критически важных данных отсутствуют;
3) время прохождение через систему известно с фиксированной точностью;
4) непосредственное взаимное влияние процессов (активностей) друг на друга отсутствует;
5) прерывания системы ввода/вывода не оказывают влияния на процессы обработки.

Многозадачность в ДСКУ основана на активных объектах A2. Активный объект – это специальная концепция Active Oberon, при которой процесс инкапсулирован в объекте [5]. Активные объекты ДСКУ создаются на фазе инициализации, имеют время жизни, равное времени жизни ДСКУ и включают в себя процедуры старта и обновления.





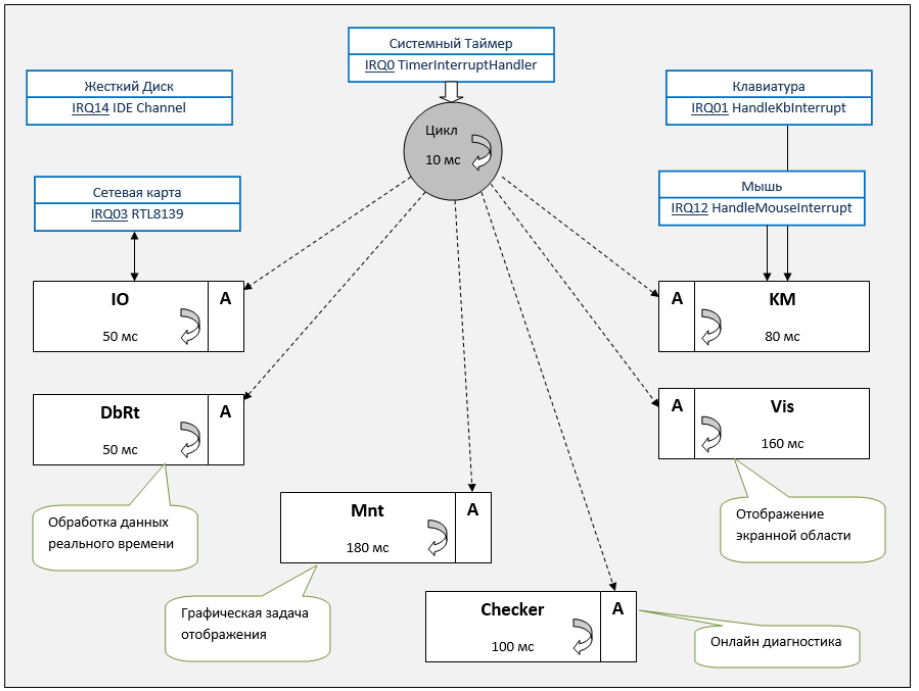


*Рис. 1. Структура активных объектов ДСКУ*
*Fig. 1. Active Objects structure of DSKU*

ДСКУ в исполнении ЧМИ состоит из следующих задач, реализуемых активными объектами (рис. 1):

- IO – ввод/вывод (в прототипе реализован как UDP с драйвером стека A2, во внедряемом варианте предполагалось использование полевой сети);
- DbRt – технологическая БД реального времени, представляющая собой массив структур статических и динамических параметров сигналов;
- KM – клавиатура и манипулятор (трекбол с заменой на touch screen во внедряемом варианте);
- Checker – онлайн диагностика состояния задач и целостности данных;
- Mnt – графическая задача отображения видеокадров и формирования команд оператора;
- Vis – формирование растра отображения экранной области.

Активные объекты изолированы, они взаимодействуют друг с другом только через объекты агентов (доказательство #2.4). Агенты осуществляют временную синхронизацию. Синхронизация обеспечивает привязку объекта к определенному фрейму вызова.

В части взаимодействия между объектами существенно, что согласно стандарту время прогона должно быть фиксированным (B.2dd, #2.3), и это должно объясняться. Поэтому взаимодействие между агентами объектов необходимо было построить по неблокирующей схеме. Это гарантирует, что объект обновляется только по временному циклу, и не ожидает наступления каких-то логических условий. Доказываются допущения #2.1, #2.2. Был выбран обмен по схеме двойной буферизации [6] (рис. 2).

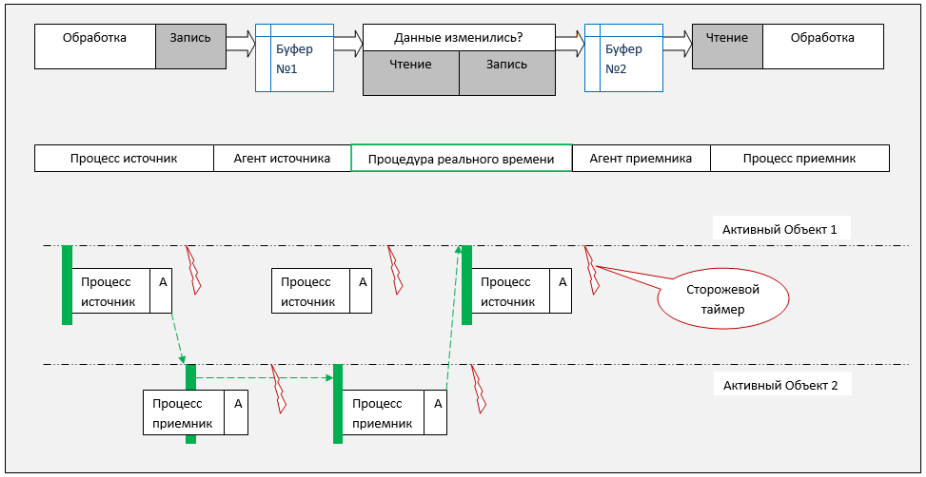


*Рис. 2. Межзадачный обмен по схеме двойной буферизации*
*Fig. 2. Double buffering inter-task communication*

Двойной буфер обеспечивает атомарный обмен области чтения на область записи. Это осуществляется процедурой реального времени, во время которой прерывания потока выполнения другими задачами запрещено.

Процессы большей длительности, например, обновление графического экрана, запускаются как задачи, разбиваемые при выполнении. При превышении времени счета задачи на 20% от установленного срабатывает сторожевой таймер, который устанавливает флаги овертайма.

Прерывания ввода/вывода ограничены по числу срабатываний за несколько фреймов вызова. При таком поведении заметна неравномерность движения трекбола. Это обеспечивает приемлемую работу манипулятора и клавиатуры. Дисковые операции запрещались после фазы инициализации (#2.5).

## 4.4 Исключения и восстановление

Исключения и восстановление соответствуют допущению #3:

1) исключения не приводят к потере ресурсов;
2) обработка исключения не приводит к генерации исключения внутри обработчика;
3) восстановление после обработки исключения всегда завершается успешно;
4) восстановление приводит к работоспособному состоянию, близкому к состоянию первого старта;
5) объект с исключением не оказывает влияние на работоспособность других объектов.

Активные объекты содержат процессы внутри себя. Конкретные объекты наследуются от базового объекта Task (рис. 3) и становятся активными. Система A2 обеспечивает инициализацию в фазе 2 вызовом конструктора &Init (…), при этом осуществляется аллокирование данных и связывание объектов с агентами. Начиная с фазы 4 происходит выполнение тела BEGIN {ACTIVE, SAFE} активного объекта в контексте процесса активного объекта, начиная с кода старта InitData. Код старта осуществляет мягкую инициализацию установкой начальных значений данных и далее по телу цикла следует фаза 5.





В фазе 5 реализован бесконечный цикл обновления данных Update по таймеру. При некорректных условиях работы программы генерируются программные исключения, относящиеся к данной задаче.

```
Task* = OBJECT
        VAR agent-: Agent; tAdvance, tBefore: INTEGER;
                statrec-: DC.Statrec; dead: BOOLEAN; ticks: LONGINT;

        PROCEDURE StartAgent* (tAdvance, tBefore: INTEGER);
        BEGIN
                …
        END StartAgent;

        PROCEDURE Start*;
        BEGIN
                …
        END Start;

        PROCEDURE Finish*;
        BEGIN
                …
        END Finish;

        PROCEDURE &Init* (agent: Agent);
        BEGIN
                ASSERT(agent # NIL, 21);
                SELF.agent := agent; statrec.errcount := -1;
        END Init;

        PROCEDURE InitData;
        BEGIN
                agent.InitData; tAdvance := 0; tBefore := 0;
                …
                Start
        END InitData;

        PROCEDURE NextTsp*;
        BEGIN {EXCLUSIVE}
        END NextTsp;

        PROCEDURE Update*;
        BEGIN
        END Update;

BEGIN {ACTIVE, SAFE}
        IF fin THEN …
        ELSE
            InitData;
            LOOP
                BEGIN {EXCLUSIVE}
                        AWAIT(fin OR (agent.tsp >= agent.start));
                        IF fin THEN dead := TRUE; EXIT END;
                END;
                …
                Update;
                …
                IF ~fin THEN
                        CASE agent.number OF
                        | 1:
                                ASSERT(agent.status # agentError, 122)
                        …
                        ELSE
                                ASSERT(agent.status # agentError, 22)
                        END
                END
            END
        END
END Task;
```

*Рис. 3. Фрагмент кода базового объекта Task*
*Fig. 3. Code snippet of the base Task object*

Семантическая конструкция {ACTIVE, SAFE} уведомляет, что данная задача имеет безопасный рестарт. При возникновении исключения при обновлении/анализе данных тело активного объекта перехватывает трап и переходит к точке рестарта.

Точка рестарта – это начало тела объекта, первый оператор которого будет код старта InitData (рис. 4). Выполняется фаза 4, за ней идет переход к фазе 5 с бесконечным циклом обновления данных. Конфигурация памяти активного объекта не меняются, происходит только смена начальных условий в InitData. Такое архитектурное решение подтверждает

корректность допущений #3.1, #3.4. Следует заметить, что данные активного объекта изолированы от других объектов и не меняются, что подтверждает доказанность независимости от других #3.5.

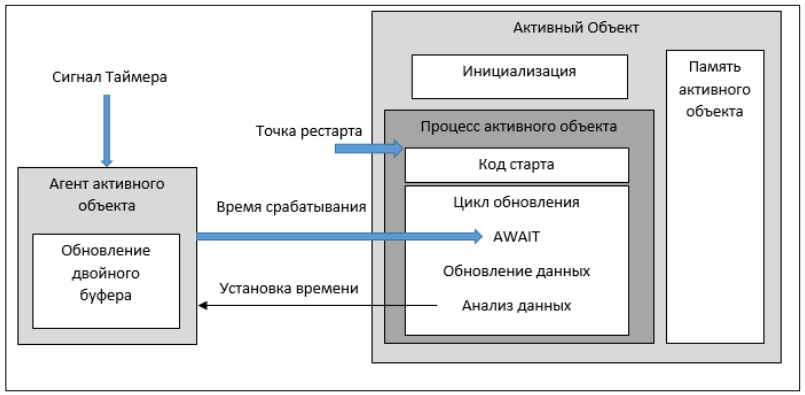


*Рис. 4. Рестарт активного объекта*
*Fig. 4. Active Object restart*

Обработчик исключений в ДСКУ не имеет внутреннего состояния и не создает новые данные. Он осуществляет обновление массивов статистики отказов для данной задаче. Таким образом, #3.2 (нет генерации исключения внутри обработчика) выполняется, а #3.3 выполняется с оговоркой, что не восстанавливаемые исключения приводят к перезагрузке системы.

В части оценки эргодичности системы при периодических восстановлениях системы (например, вызванных овертаймами) можно утверждать, что каждое новое восстановленное состояние идентично предыдущему с точностью до временных меток.

## 4.5 Отсутствие потерь информации при сетевых сбоях

В качестве допущений #4 корректной работы сети используются сформулированные в [7]:

1) сетевая передача является надежной;
2) латентность равна 0 или не учитывается;
3) пропускная способность бесконечна;
4) транспортные издержки равны 0 или не учитываются.

Согласно стандарту 60800, «*B.2dg:* объем данных, считываемых в течение одного вычислительного цикла, должен быть постоянным». Поэтому по сети осуществляется циклическая передача всех данных фиксированного размера с циклом обновления 10-50 мс. При этом передача осуществляется по дублированной сети. Даже при наличии многократных сетевых сбоев информация приходит в ДСКУ со следующим циклом, т.е. с некоторой задержкой, кратной периоду обмена.

Для прототипа использовался протокол UDP. Использование «эластичных» протоколов, в частности, TCP ухудшает свойства эргодичности при сетевой передаче, т.к. соединение может рваться и не восстанавливаться при сложных условиях эксплуатации.

Циклический прием данных обеспечивает надежное #4.1 гарантированное считывание порции данных за 2 сетевых цикла, при условии, что произошло не более 1 сбоя хотя бы в одной из 2 сетевых карт. Латентность #4.2 и транспортные издержки #4.4 не учитываются, т.к. ДСКУ не предназначена для работы через радиоканалы. Проблемы с пропускной способностью #4.3 тоже нет, т.к. при постоянной передаче фиксированного размера





максимальная пропускная способность является номинальной. И на этапе отладки с номинальным потоком сигналов пропускная способность легко тестируется.

## 4.6 Отсутствие переполнений времени и проблемы 2038

Проблемы с переполнением времени соответствуют допущению #5:

1) отсутствуют переполнения счетчиков времени;
2) величина будущей временной метки всегда больше текущей;
3) нет скрытых блокировок ожидания.

Существует множество известных проблем переполнения времени (проблемы 2038, 2106). Они связаны с переходом через 0 целочисленного счетчика времени. Последствия такой неэргодичности могут быть катастрофическими, например, приложения, использующие функцию Linux nanosleep, для многих ядер зависают при переходе 19 января 2038 года.

Единственный таймер ДСКУ не привязан к текущему времени или временным зонам и обновляет 64-разрядный счетчик tsp с момента начала загрузки, а не начального момента времени 1900 или более раннего года. Утверждения #5.1, #5.2 доказаны, т.к. нет и не будет оборудования, способного непрерывно отработать $10^8$ лет при цикле 1 мс. В системе нет понятия текущего времени. Задержки реализуются только обратным отсчетом установленных начальных значений. Такая схема гарантирует отсутствие переполнений (проблемы 2038, 2106). Системы синхронизации и корректировки времени типа NTP не используются ДСКУ, и в целом для категории А. Следовательно, блокировок ожидания #5.3 нет.

## *5. Недостатки предлагаемого подхода*

Повышение надежности и переход системы в стабильно стационарное состояние имеет свою обратную сторону. Процессорное время будет использовано гораздо менее эффективно. Выполнение требований стандарта исключает циклы вида WHILE DO, REPEAT UNTIL, которые не имеют постоянного числа итераций. Даже алгоритм линейного поиска выполняется циклом FOR без возможности преждевременного завершения при успешном окончании. Входные события, например, изменение состояний дискретного сигнала, обрабатываются не как события, а как весь массив входных данных.

Увеличение интенсивности прерываний наблюдалось при замене трекбола на манипулятор-мышь, при интенсивной работе которое наблюдались рывки. Это объяснялось ограничением числа прерываний за цикл обмена задачи.

Следует заметить, что применение данных систем необходимо при реализации простых алгоритмов в части защит, блокировок, управления АСУ ТП. Более сложные алгоритмы не имеют таких требований в части функциональной безопасности.

С точки зрения аппаратного обеспечения работа с постоянной стабильной нагрузкой на процессор будет даже предпочтительным решением, а современные мощности позволят все и далее переходить ко все более надежным решениям.

## *6. Презумпция неэргодичности*

Вопрос наличия свойства эргодичности возникает для многих систем, предназначенных к долгосрочной эксплуатации. В большинстве случаев разработчики не проектируют заранее систему с учетом указанных свойств, а реагируют на возникновении конкретных технических проблем, пока это не станет носить системный характер. При этом программная система постулируется работоспособной, если она успешно функционирует сразу после загрузки.

Презумпция неэргодичности предполагает, что данное свойство отсутствует, и разработчику необходимо обосновать эргодичность системы, в том числе и рассмотрением архитектурных особенностей. Оберон-системы на основе A2 могут стать хорошим фундаментом для построения. Представленная система ДСКУ обладает доказательствами эргодичности в части:

- управления памятью;
- обеспечения реального времени;
- механизма обработки исключений;
- сетевого обмена данными;
- обеспечения управления временем.

Данный список может быть расширен при рассмотрении требований конкретных промышленных стандартов, которые не принимались во внимание, например, Авионики. Но тенденция сохраняется.

### *7. Заключение: развитие от отрицания «via negativa»*

Знание, от каких технологических решений следует отказаться, часто более ценно, чем знание, какие технологические решения следует применять. Механизмы, приводящие к улучшению свойств эргодичности, основаны на ограничениях и запретах:

1) предсказуемость работы с памятью обеспечивается отказом от виртуальной подкачки страниц в пользу более простой прямой адресации;
2) гарантии управления памятью основаны на запрете динамического выделения после фазы инициализации;
3) жесткое реальное реализуется отказом от ОС на «голом железе»;
4) межзадачный обмен осуществляется только одним допустимым способом через двойную буферизацию. Другие способы исключены;
5) при рестарте запрещена инициализация, только программный код старта;
6) при сетевой передаче допускается только обмен посылками фиксированной длины;
7) установка времени и синхронизации времени запрещена.

Обобщая эти факторы, можно сделать вывод, что отказ от большинства используемых технологий приводит совершенно новым качествам, недостижимым путем добавления каких-либо еще более изощренных программных решений.

И, что важно, построенная на таких аскетических принципах программная система не только обладает свойствами эргодичности – она также становится объяснимой и доказуемой.

## Список литературы / References

[1]. O. Peters, M. Gell-Mann. Evaluating gamples using dynamics. Chaos: An Interdisciplinary Journal of Nonlinear Science, vol. 26, issue 2, 2016, article id 023103.
[2]. O. Peters. The ergodicity problem in economics. Nature Physics, vol. 15, issue 12, 2019, pp. 1216-1221.
[3]. Соловьева А.А., Цапкова Н.Н., Покровский В.И. Будущее принадлежит медицине предохранительной – Н.И.Пирогов. Терапевтический архив, том 83, no. 11, 2011 г., стр. 5-9 / Solovyeva A.A., Tsapkova N.N., Pokrovsky V.I. "Medicine of the future is preventive medicine" N.I. Pirogov. Therapeutic archive, vol. 83, no. 11, 2011, pp. 5-9 (in Russian).
[4]. ГОСТ Р МЭК 60880, Программное обеспечение компьютерных систем, выполняющих функции категории А, 2009 / GOST R IEC 60880, Software for computer systems performing category A functions, 2009 (in Russian).
[5]. Pieter J. Muller, The Active Object System Design and Multiprocessor Implementation. Diss. ETH No. 14755, for the degree of Doctor of Technical Sciences, ETH Zurich 2002, 197 p.





[6]. S. Louise, M. Lemerre, C. Aussagues and V. David. The OASIS Kernel: A Framework for High Dependability Real-Time Systems. In Proc. of the IEEE 13th International Symposium on High-Assurance Systems Engineering, 2011, pp. 95-103.
[7]. Arnon Rotern-Gal-Oz. Fallacies of Distributed Computing Explained. URL: https://www.researchgate.net/publication/322500050_Fallacies_of_Distributed_Computing_Explained, accessed 20.11.2020.

## Информация об авторе / Information about the author

Дмитрий Викторович ДАГАЕВ – главный эксперт «Русатом – Автоматизированные системы управления», консультант проекта «Информатика-21». Сфера научных интересов: разработка системного ПО АСУТП, разработка кросс-платформенных Оберон-технологий.

Dmitry Victorovich DAGAEV – Chief Expert of Rusatom – Automated Control Systems JSC, consultant of the Informatika-21 project. Research interests: DCS system software development, cross-platform Oberon technologies investigation.