ISSN 1992-6502 (Print) Вестник УГАТУ ISSN 2225-2789 (Online)
2025. Т. 29, № 2 (108). С. 77-88 http://journal.ugatu.su

УДК 004.451.42, 004.451.47 doi 10.54708/19926502_2025_29210877
Код ГРНТИ 20.53.17, 20.53.19, 20.53.23

Многоагентный подход к организации программного обеспечения в киберфизических системах

А.С. Ковтуненко*, Д.Е. Ваганова, П.К. Серебряков

ФГБОУ ВО «Уфимский университет науки и технологий», г. Уфа, Россия

Аннотация. В работе рассматривается проблема организации программного обеспечения в киберфизических системах (КФС). Предлагается концепция, согласно которой основными вычислительными процессами в КФС являются получение и обработка потоковых данных. Предложена математическая модель распределенной обработки потоковых данных в киберфизических системах, которая отражает основные требования к программному обеспечению КФС: гибкость и живучесть. Предложенная модель реализована в виде программной архитектуры, позволяющей существенно повысить эффективность жизненного цикла программного обеспечения для КФС.

Ключевые слова: *киберфизические системы, программная архитектура, потоковые данные, многоагентные системы, программный агент.*

***askovtunenko@mail.ru**

Введение

Под киберфизической системой традиционно понимается информационно-технологическая концепция, подразумевающая интеграцию вычислительных ресурсов в физические сущности любого вида, включая биологические и рукотворные объекты. В киберфизических системах вычислительные задачи распределены по всей физической системе, соответственно, вычислительные ресурсы для реализации функционала могут включать в себя ресурсы как узлов общего назначения (автоматизированных рабочих мест, серверов хранения и обработки данных и пр.), так и специализированных аппаратных средств (аппаратных платформ сбора данных и управления: актуаторов и шлюзов датчиков, маломощных вычислительных устройств и пр.). С точки зрения вычислительной системы аппаратный слой киберфизических систем можно охарактеризовать как гетерогенную вычислительную сеть с непредсказуемой загрузкой.

Таким образом, подход к рассмотрению обработки данных в распределенных киберфизических системах должен быть гибким, а также подразумевать открытость для интеграции различных архитектур между собой.

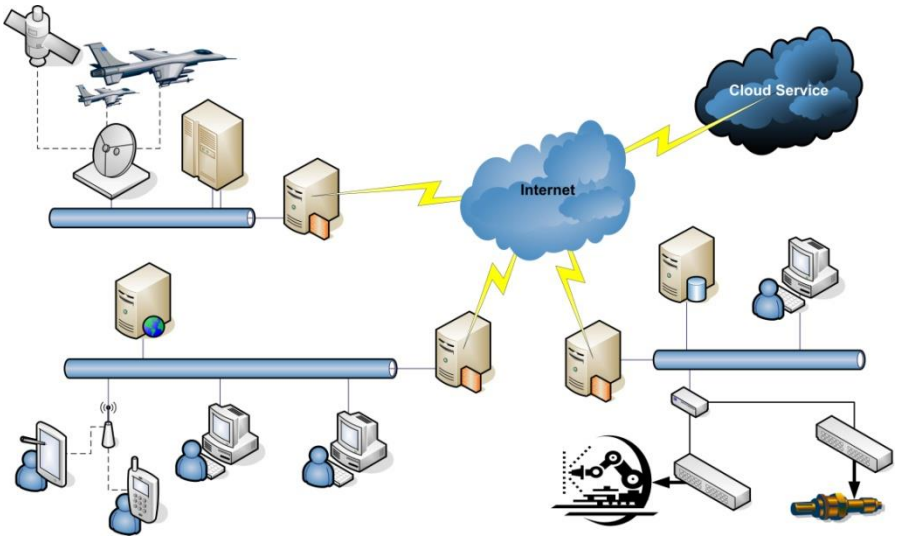


Рисунок 1. К пониманию киберфизической системы.

В состав аппаратного обеспечения киберфизической системы могут входить устройства и технические решения совершенно разнообразной природы и ресурсоемкости (Рис. 1). Поскольку киберфизическая система, как правило, является частью более обширной организационно-технической системы, к алгоритмическому и программному обеспечению предъявляются особые требования по согласованности функционирования, эффективности взаимодействия всех подсистем, унификации моделей представления вычислительных и иных ресурсов, а также функционала системы [1].

Концепция обработки данных в киберфизических системах

В информатике как отрасли науки проблематике данных посвящено множество теоретических и прикладных исследований, а также научных направлений. Примечательно, что само понятие «информация» редко кем из исследователей определяется однозначно. Академик Н.Н. Моисеев даже полагал, что в силу широты этого понятия нет и не может быть строгого и достаточно универсального определения. Не погружаясь в риторику о понятии «информация», будем рассматривать ее в дихотомии, то есть разделяющейся на две категории: данные и знания. Под данными будем понимать информацию о фактах (событиях, явлениях) окружающего мира, а под знаниями – информацию о взаимосвязи данных (фактов) между собой (зависимостях, закономерностях).

Применительно к киберфизическим системам данные можно разделить на два типа по специфике их хранения и обработки: потоковые и событийные данные [2].

Событийные данные несут преимущественно информацию о наступивших в физической среде событиях, фактом наступления которых нельзя пренебречь. Такие данные чаще всего представляют собой логический индикатор (флаг события), который вместе с отметкой времени обрабатывается системой, причем эффективность обработки однозначно зависит от факта обработки каждого события, поскольку, если информация о событии будет в процессе обработки по каким-то причинам утеряна, система не выполнит одну из своих функций [3].

Потоковые данные, в свою очередь, несут информацию о состоянии некоторого физического явления или процесса и чаще всего представляют собой числовое значение какого-либо параметра (свойства) физической среды. Вместе с отметкой времени (момента регистрации значения на границе киберфизической системы) значение должно быть обработано системой за установленное время. Таким образом, потоковым данным сопутствует понятие «актуальности», что означает, в случае, когда значение по каким-либо причинам не может быть обработано в установленное время, оно теряет «актуальность» и может быть отброшено без ущерба для выполнения функции системы. В качестве примера таких данных можно привести наблюдение за текущим состоянием окружающей среды с целью предупреждения потерь от природных катаклизмов [4].

В контуре киберфизической системы происходит получение и преобразование разных типов данных между собой и друг в друга (Рис. 2).

Другим важным аспектом при рассмотрении программно-аппаратных систем являются ресурсы. При рассмотрении киберфизических систем выделим следующие виды вычислительных ресурсов:

– оперативная память (в байтах);

– производительность (во флопсах);

– объем сетевого взаимодействия (в байтах за секунду).

Специфика функционирования киберфизических систем требует дополнительного выделения следующих видов ресурсов:

– канал взаимодействия с физической средой: датчики, актуаторы (используется/не используется);

– электроэнергия (кВт/ч).

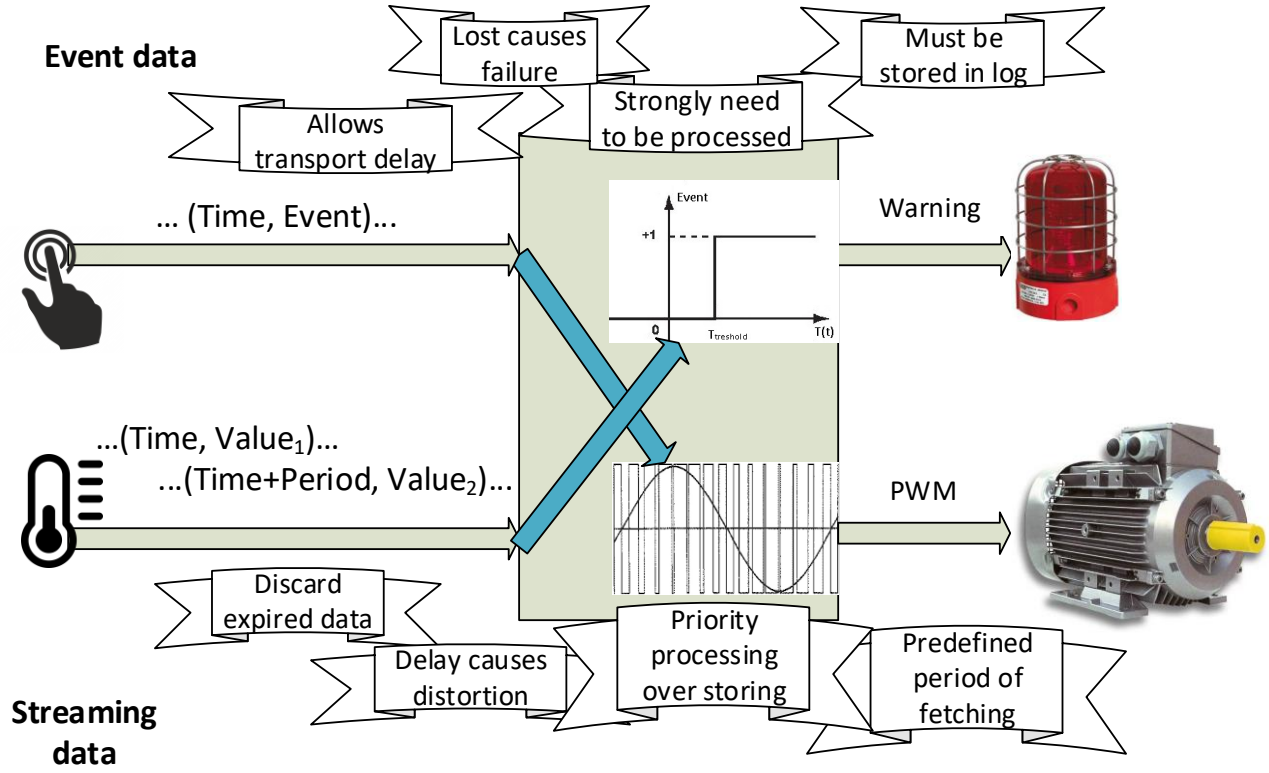


Рисунок 2. К пониманию дихотомии потоковых и событийно-ориентированных данных в КФС.

Таким образом может быть сформулирована следующая типология вычислительных процедур и, как следствие, программных компонент.

Таблица 1. Типология вычислительных процедур

№	Тип процедуры	Данные на входе	Данные на выходе	Ресурс
1.	Получение (регистрация) данных	Физическая среда	Потоковые, Событийные	Датчик Память
2.	Преобразование потоковых данных	Потоковые	Потоковые	Память Производительность Сеть
3.	Генерация простых событий	Потоковые	Событийные	Память Производительность Сеть
4.	Обработка событий (генерация/обработка сложных событий)	Событийные	Событийные	Память Производительность Сеть
5.	Генерация задающих сигналов	Событийные	Потоковые	Память Производительность Сеть
6.	Отправка (утилизация) данных	Потоковые, Событийные	Физическая среда, память	Память, Актуатор

Например, при получении первичных данных программная компонента использует ресурс датчика (подключенного, например, по интерфейсу UART) или дискретного входа (интерфейс GPIO) и сохраняет данные в памяти. Например, программная компонента косвенных измерений реализует запрограммированный алгоритм и по нескольким текущим значениям разных показателей рассчитывает новое значения вычислимого показателя, которое также сохраняется в памяти. Программная компонента генерации простых событий сопоставляет значение какого-то потокового параметра с пороговым значением и при превышении генерирует событие «превышение порога», и так далее. Программная компонента логирования сохраняет все значения какой-либо характеристики в базе данных, реализуя тем самым утилизацию данных (Рис. 3).

Существенная особенность современных подходов к построению программных средств для КФС заключается в том, что в качестве базового типа данных негласно подразумеваются события. Поэтому и инструментарий, и программные архитектуры, и промежуточное ПО являются событийно- или сообщение-ориентированными. Все потоковые данные искусственно притягиваются к дискретно-событийной природе. То, что называется «потоками данных», является потоковыми данными в синтетической форме. Это влечет за собой нерациональное использование ресурсов, усложнение программных систем для обеспечения устойчивости функционирования и прочие негативные эффекты, которых можно было бы избежать.

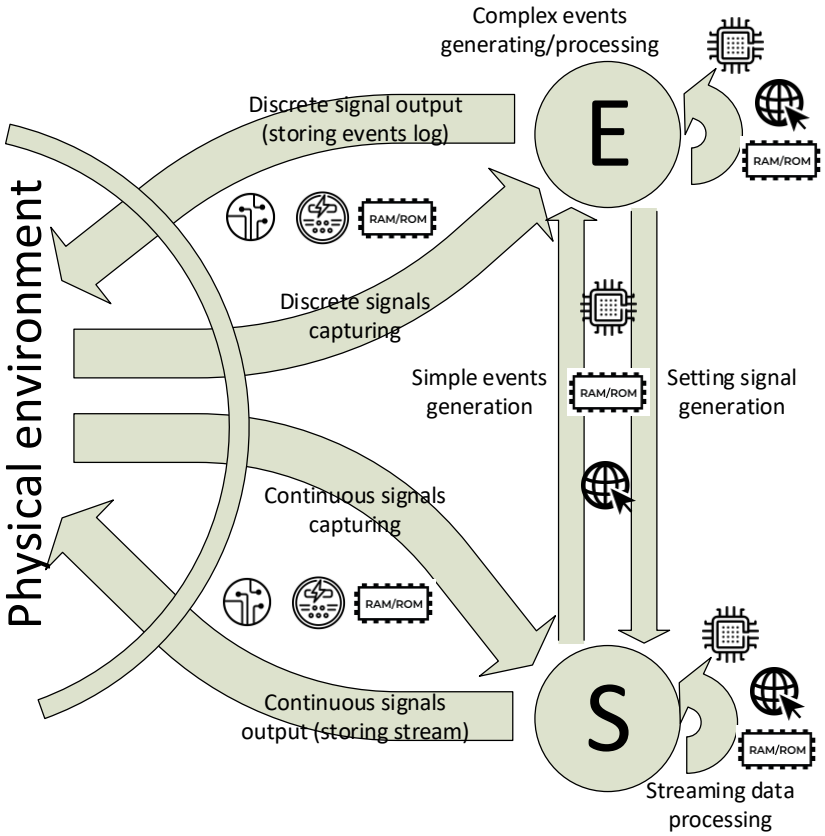


Рисунок 3. Процедуры преобразования данных в КФС.

Представленный подход подразумевает рассмотрение в качестве базового типа потоковые данные, что наилучшим образом характеризует специфику обработки информации в киберфизических системах, и при проектировании программного и алгоритмического обеспечения позволяет структурировать функционал системы.

Математическая модель обработки потоковых данных в распределенных киберфизических системах

Атомарным элементом потоковых данных в рамках предлагаемой модели является *атрибут*, который соответствует переменной простого типа, представляющей какую-либо физическую величину в системе. С каждым атрибутом связана вычислительная *процедура* (программная компонента), которая с заданной периодичностью циклически получает новое значение атрибута путем взаимодействия с физической средой или путем выполнения вычислений с использованием значений других атрибутов [5].

Модель данных, таким образом, представляет собой множество атрибутов $\boldsymbol{A}$ и вычислительных процедур $\boldsymbol{P}$, связанных между собой следующими отношениями.

– Отношение, которое задает какая вычислительная процедура рассчитывает значения каждого атрибута:

$$C : A \to P,$$
$$C = \{(a, p) : a \text{ вычисляется процедурой } p, a \in A, p \in P\} \quad (1)$$

так, что $C(a)$ – процедура, которая вычисляет значения атрибута a.

– Отношение, которое определяет, какие атрибуты используются для расчетов в каждой процедуре так, что $C \cap I = \varnothing$,

$$I : P \to A,$$
$$I = \{(p, a) : p \text{ использует } a, p \in P, a \in A\} \quad (2)$$

так, что $I(p) \subseteq A$ – атрибуты, значения которых используются в расчетах в процедуре p.

В рамках настоящей модели рассмотрим вычислительные ресурсы только трех основных типов:

– оперативная память;

– производительность процессора;

– объем сетевого взаимодействия.

Применительно к модели данных ресурсоемкость задачи обработки потоковых данных может быть представлена следующими отношениями.

$$M : A \to N \setminus \{0\}$$

так, что $M(a)$ – объем памяти, необходимый для хранения одного значения атрибута a в байтах.

$$V : P \to N \setminus \{0\}, \quad (4)$$

так, что $V(p)$ – вычислительный объем процедуры p в количестве операций с плавающей точкой.

$$T : P \to R^{+}, \quad (5)$$

так, что $T(p)$ – установленный период повтора процедуры p в секундах.

Аппаратное обеспечение киберфизической системы в общем случае представляет собой гетерогенную вычислительную сеть с непредсказуемой загрузкой. Рассмотрим ее как совокупность вычислительных узлов, образующих полносвязную сеть, каждый из которых имеет запас вычислительных ресурсов.

Введем множество вычислительных узлов $\boldsymbol{W}$, на которых физически исполняются все вычислительные процедуры $\boldsymbol{P}$, причем размещение процедур на узлах задается следующим отношением:

$$D = \{(p, w) : p \; p \text{ исполняется на узле } w, p \in P, w \in W\}, \quad (6)$$

так, что $D(p)$ – вычислительный узел, на котором физически исполняется процедура p.

Иными словами, вычислительные ресурсы узла *D(p)* используются процедурой *p* для хранения атрибутов $C^{-1}(p)$, для проведения с периодичностью *T(p)* расчетов по актуализации их значений, а также для получения значений атрибутов *I(p)*, которые, возможно, расположены физически на других узлах.

Таким образом, ресурсная модель вычислительной сети определяется следующим образом:

$$\begin{aligned} &\forall w \in W, \\ &S_{MEM} : W \to N, \\ &S_{CPU} : W \to R, \\ &S_{NET} : W \to R, \end{aligned} \tag{7}$$

так, что:

$S_{MEM}(w)$ – количество доступной оперативной памяти на узле *w*, байт;

$S_{CPU}(w)$ – максимально достижимая производительность на узле *w*, флопс;

$S_{NET}(w)$ – максимально достижимая скорость получения данных по сетевому интерфейсу для входящего трафика, байт/секунду.

Используя предложенную математическую модель, можно сформулировать критерий реализуемости поставленной задачи обработки потоковых данных на базе существующей вычислительной сети киберфизической системы.

Поскольку размещение процедур по вычислительным узлам определяется отношением *D*, размещение считается допустимым, а задача обработки потоковых данных реализуемой в том случае, если вычислительные узлы имеют достаточно ресурсов для проведения расчетов с заданной периодичностью.

Для этого необходимо, чтобы $\forall w \in W$ выполнялись все нижеперечисленные неравенства:

$$\begin{aligned} &\sum_{\forall a \in (D^{-1} \circ C^{-1})(w)} M(a) < S_{MEM}(w), \\ &\sum_{\forall p \in D^{-1}(w)} \frac{V(p)}{T(p)} < S_{CPU}(w) \\ &\sum_{\forall p \in D^{-1}(w)} \frac{\sum_{\forall a \in I(p) \setminus (D^{-1} \circ C^{-1})(w)} M(a)}{T(p)} < S_{NET}(w) \end{aligned} \tag{8}$$

Предложенный подход позволяет гибко создавать математические модели различных типов вычислительных сетей и задач обработки данных и формулировать разнообразные задачи управления ресурсами киберфизической системы [6].

Обобщенная архитектура программного обеспечения киберфизической системы

В работе предлагается рассматривать подсистему обработки данных киберфизической системы как совокупность следующих элементов (Рис. 4):

– гетерогенная полносвязная вычислительная сеть;

– общая программная платформа на всех узлах сети;

– общее хранилище для программных компонентов КФС;

– распределенная среда исполнения, включающая интерпретаторы императивных и декларативных ЯП;

– ПО КФС в виде исполняемых взаимодействующих компонентов.

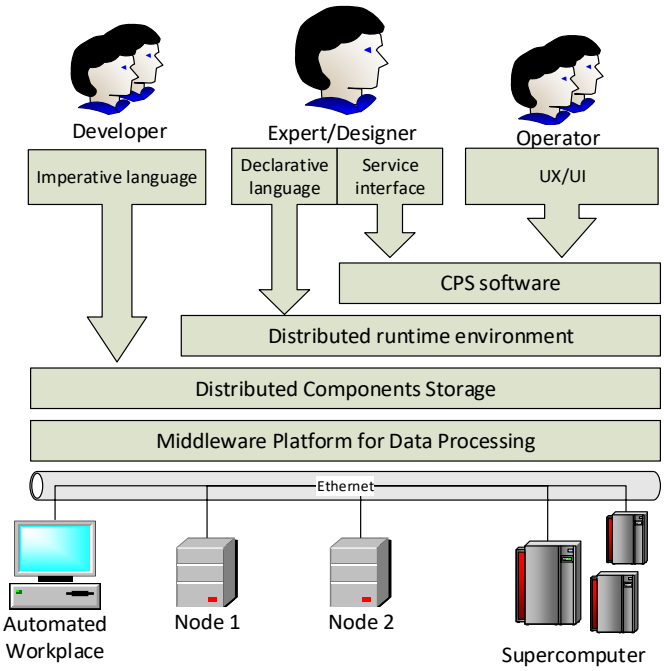


Рисунок 4. Обобщенная архитектура программного обеспечения КФС.

Гетерогенная вычислительная сеть, как было отмечено ранее, является основным источником вычислительных ресурсов в КФС и включает в себя устройства, различные по вычислительной мощности, назначению, используемым аппаратным и программным платформам. Для наглядности можно представить, что сеть является одноранговой полносвязной сетью передачи данных на базе Ethernet [7].

Предлагаемая обобщенная архитектура подразумевает, что на каждом узле вычислительной сети запущен исполняемый компонент платформы распределенной обработки данных, а также исполняемый модуль распределенного хранилища программных компонент (подпрограмм, классов, модулей). Жизненный цикл системы включает разработку компонентов системы и размещение их общем хранилище компонентов. Разработчик алгоритмов использует для этого один из императивных языков программирования (таких как Java, C/C++, Python). Каждый компонент соответствует одной процедуре обработки данных p – элементарной функции системы. С применением декларативного языка эксперт-дизайнер конструирует из этих компонентов описание всей системы, определяя конкретные конфигурации компонентов, а также связи между ними. На основании этого описания служебные компоненты платформы создают систему как набор экземпляров компонент, созданных и сконфигурированных в соответствии с представленным описанием. После запуска программного обеспечения оператор системы может взаимодействовать с ней посредством графического интерфейса пользователя или операторского пульта управления.

Данная архитектура позволяет повысить гибкость создаваемых программных систем за счет достаточно простой реконфигурируемости, а также повысить эффективность жизненного цикла программного обеспечения CPS за счет разделения реализации алгоритмов и концептуального проектирования.

Программная архитектура организации обработки потоковых данных в распределенных CPS

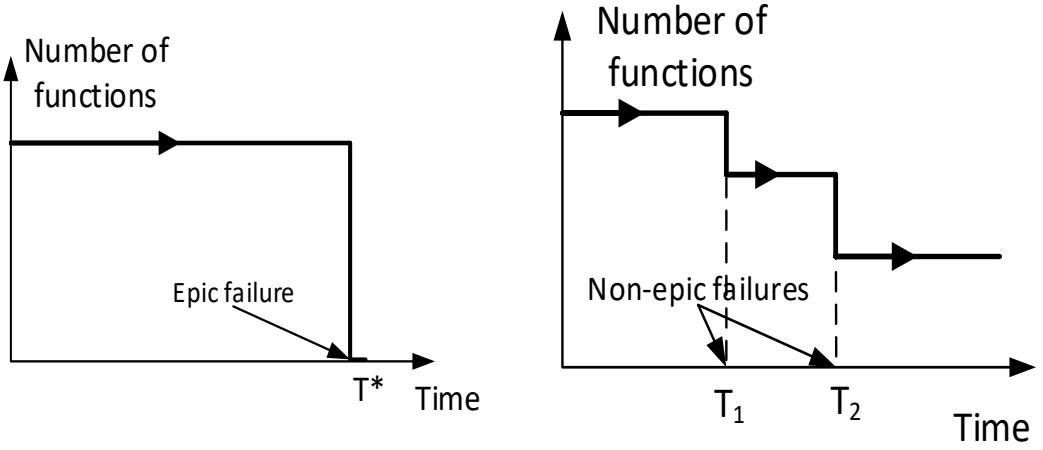
Одним из самых значимых требований к программному обеспечению КФС является, на наш взгляд, требование функциональной устойчивости [8], или, иначе. функциональной безопасности [9]. Обобщенно функциональную устойчивость программной системы можно определить как способность обеспечивать выполнение функций максимально длительное время.

На это оказывает влияние множество разнообразных факторов, характеризующих систему, однако традиционно рассматривается два подхода к пониманию функциональной устойчивости: отказоустойчивость и живучесть.

Под обеспечением отказоустойчивости системы будем понимать комплекс мероприятий, направленный на сохранение полного функционала системы в условиях возникновения нештатных ситуаций. Иными словами, система должна максимально долго сохранять возможность выполнять все свои функции (Рис. 5а). С другой стороны, свойство живучести программной системы подразумевает сохранение ограниченного функционала системы в условиях отказа отдельных компонентов (Рис. 5б). Обеспечение живучести системы подразумевает использование в качестве базы для проектирования функционально слабосвязанных программных компонент.

Очевидно, что живучесть программной системы для функционирования КФС играет более важное значение. В настоящее время наиболее точно отвечающей принципам построения живучих систем является концепция агентно-ориентированного подхода.

Как развитие обобщенной архитектуры в настоящей работе предложена также программная архитектура систем распределенной обработки потоковых данных в киберфизических системах. Она относится к одноранговым и реализует основные принципы агентно-ориентированного подхода к построению программного обеспечения [10].



(а) (б)

Рисунок 5. К понятию отказоустойчивости:
(а) критическое событие; (б) некритические события.

В рамках предлагаемой агентно-ориентированной программной архитектуры программная система рассматривается как совокупность автономных программных компонент, взаимодействующих между собой в едином пространстве имен в режиме максимально приближенном к реальному времени. К основным элементам архитектуры можно отнести следующие (Рис. 5):

– In-memory распределенная база данных, которая представляет собой быстродействующее key-value хранилище, используемое для хранения и обеспечения доступа к значениям атрибутов *a*.

– Агентная платформа – единая распределенная программная среда, в которой выполняются и взаимодействуют элементарные программные компоненты обработки данных – программные агенты (процедуры p). Они вместе и реализуют требуемый прикладной функционал системы. Представляет собой совокупность агентных контейнеров.

– Агентные контейнеры – драйвера доступа к физическим вычислительным ресурсам узла w. Если агентная платформа – это абстрактная распределенная структура, то контейнер – экземпляр программной компоненты, запущенный на физическом вычислительном узле

и обеспечивающий компонентам более высокого абстрактного уровня – программным агентам – доступ к памяти, процессору и прочим ресурсам узла сети.

– Программные агенты – автономные программные компоненты, исполняемые в едином пространстве имен – агентной платформе, однако функционально привязанные к физическим узлам – агентным контейнерам. В рамках предлагаемой программной архитектуры могут быть двух типов:
целевые агенты – однозначно соответствующие процедурам обработки потоковых данных *p*;
служебные агенты – компоненты, выполняющие вспомогательные функции, обеспечивающие основные функции платформы.

– Язык описания систем обработки потоковых данных (streaming data processing systems description language – SDP SDL) – основанный на XML декларативный язык-протокол описания спецификаций обработки данных, размещения и конфигурации и прочих параметров программной системы. Позволяет повысить эффективность проектирования, построения и развертывания распределенной программной среды обработки данных (Рис. 6) за счет разделения процесса реализации процедур обработки данных и концептуального проектирования и построения системы.

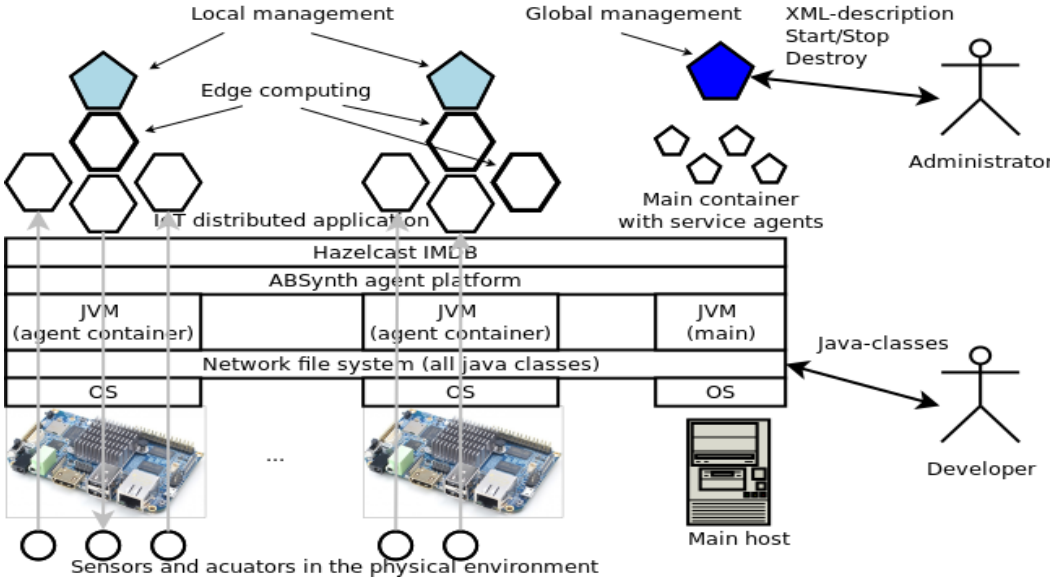


Рисунок 5. Агентно-ориентированная программная архитектура распределенной обработки потоковых данных.

Кроме перечисленных выше основных элементов программная архитектура включает ряд алгоритмических разработок по управлению ресурсами гетерогенных вычислительных сетей [11]. Рассмотрение их выходит за рамки данной работы.

ABSynth: Агентно-ориентированный фреймворк для распределенной обработки данных

Предложенная агентно-ориентированная программная архитектура распределенной обработки потоковых данных реализована на языке Java в виде агентно-ориентированного фреймворка ABSynth, функционал которого непрерывно расширяется и дорабатывается. В настоящее время он включает в себя два крупных пакета.

Пакет «платформа» включает запускаемую компоненту-движок – класс absynth.Platform, каждый запущенный экземпляр которого по сути является агентным контей-

нером. Поскольку фреймворк ABSynth в свою очередь использует функционал более глобального фреймворка JADE, обязательным является запуск в первую очередь главного контейнера, который содержит ряд специализированных служебных агентов, обеспечивающих функционирование агентной платформы. При запуске остальных (периферийных) контейнеров, обязательным является указание в параметрах запуска ip-адреса главного контейнера. Также в данном пакете описаны основные типы агентов:

– Главный менеджер – присутствует в единственном экземпляре на главном контейнере и осуществляет создание и запуск всей распределенной программной системы обработки потоковых данных. Выполняет парсинг SDP SDL файлов.

– Периферийный менеджер – создается в единственном экземпляре на каждом периферийном контейнере в момент его запуска, осуществляет ретрансляцию служебных команд (например, запуска выполнения, перемещения или уничтожения) целевым агентам своего контейнера, а также осуществляет мониторинг ресурсов своего вычислительного узла. Кроме того, реализует интерфейс IMDB Hazelcast, который используется для хранения и передачи атрибутов, который предоставляет своим целевым агентам.

– Целевой агент – путем реализации полиморфизма может быть сконфигурирован для решения задачи любой вычислительной процедуры p модели обработки потоковых данных. Через интерфейс менеджера своего узла имеет доступ в IMDB, где может получать значения любых атрибутов.

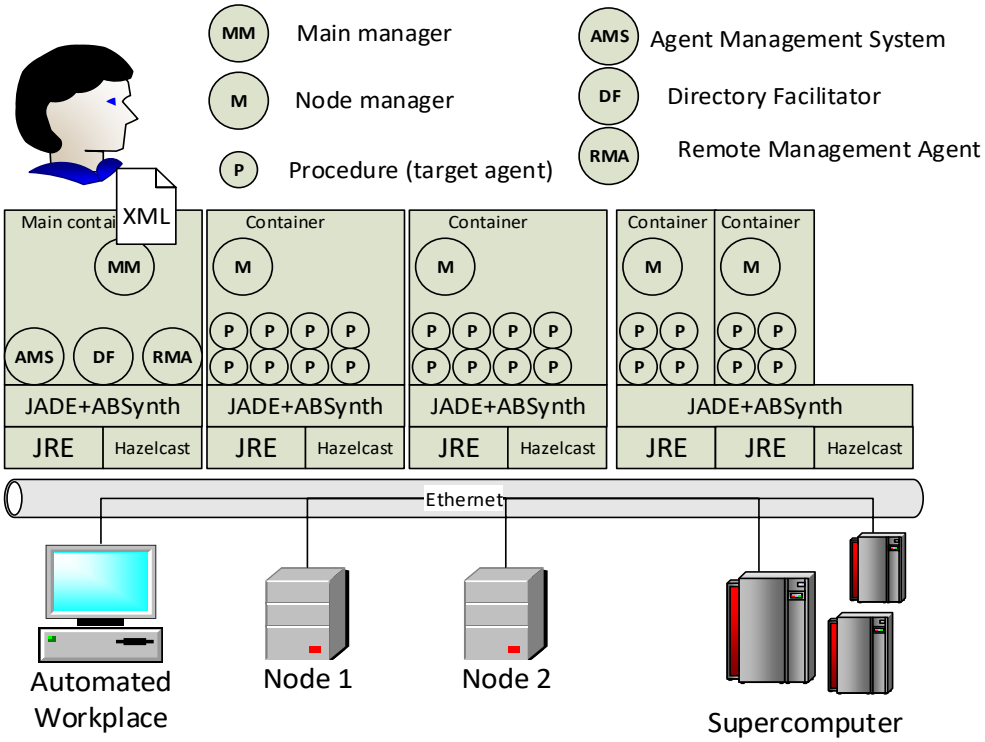


Рисунок 6. Структура распределенной платформы.

Вторым большим пакетом в составе фреймворка является пакет «библиотека», который включает в себя классы-наследники класса «целевой агент». Они реализуют различный функционал общего назначения – подпакет «common», а также специализированного – подпакет «model».

В целом фреймворк ASBynth является законченным программным продуктом, который может быть использован для организации программного слоя обработки данных в киберфизических системах при полунатурном моделировании или прототипировании [12].

Заключение

– В работе предложен подход к рассмотрению КФС в качестве гетерогенной вычислительной компьютерной сети, предназначенной для накопления и обработки потоковых данных.

– Приведена математическая модель обработки потоковых данных.

– На основе математической модели сформулирована проблема потребления ресурсов в гетерогенных вычислительных сетях.

– Разработана обобщённая архитектура КФС для распределённых систем.

– Разработано агентно-ориентированное ПО с прицелом на повышение живучести КФС. ПО реализовано как Java фреймворк ABSynth и может быть использовано для разработки под распределённые КФС.

Литература:

1. Родионова Л.Е., Антонов В.В., Баймурзина Л.И., Гидинда Г.М. Модели проектирования программных аналитических комплексов с декартово замкнутой категорией // Системная инженерия и информационные технологии. 2023. Т. 5. № 5(14). С. 3–15. [Rodionova L.E., Antonov V.V., Baimurzina L.I., Gidinda G.M. Models for designing software analytical systems with a Cartesian closed category // Systems Engineering and Information Technologies. 2023. Vol. 5. No. 5(14). P. 3–15 (in Russian)]. DOI 10.54708/2658-5014-SIIT-2023-no5-p3. EDN AQLGLE.

2. Discrete Event Simulations / Ed. by A. Goti. Rijeka: SCIYO Publ., 2010. DOI: 10.5772/257

3. Ferscha A. Parallel and Distributed Simulation of Discrete Event Systems. Parallel and Distributed Computing Handbook. McGraw-Hill, 1996.

4. Воробьев А.В., Воробьева Г.Р., Христодуло О.И. Программная система пространственной визуализации прогностических и ретроспективных данных вероятности наблюдения полярных сияний // Научно-технический вестник информационных технологий, механики и оптики. 2021. Т. 21. № 2. С. 225–233. [Vorobev A.V., Vorobeva G.R., Khristodulo O.I. An information system for spatial visualization of prognostic and retrospective data on the probability of observing auroras // Scientific and Technical Journal of Information Technologies, Mechanics and Optics. 2021. Vol. 21. No. 2. P. 225–233 (in Russian)]. DOI: 10.17586/2226-1494-2021-21-2-225-233.

5. Volkov A., Al-Sveiti M., Elgendy I.A., Kovtunenko A.S., Muthanna A. Detection and recognition of moving biological objects for autonomous vehicles using intelligent edge computing/LoRaWAN mesh system // Lecture Notes in Computer Science. 2020. Vol. 12526. P. 3–15. DOI: 10.1007/978-3-030-65729-1_1. EDN OYNAME.

6. Ковтуненко А.С., Тимиров М.А., Валеев С.С. Управление ресурсами системы, распределенной обработки потоковых данных на основе многоагентного подхода // Естественные и технические науки. 2018. № 10(124). С. 179–181. [Kovtunenko A.S., Timirov M.A., Valeev S.S. Resource management of a distributed processing system of streaming data based on a multi-agent approach // Natural and Technical Sciences. 2018. No. 10(124). P. 179–181 (in Russian)].

7. Tchernykh A., Bychkov I., Feoktistov A., Gorsky S., Sidorov I., Kostromin R., Edelev A., Zorkalzev V., Avetisyan A. Mitigating uncertainty in developing and applying scientific applications in an integrated computing environment // Programming and Computer Software. 2020. Vol. 46. No. 8. P. 483–502. DOI: 10.1134/S036176882008023X. EDN EVUUTF.

8. Ankica Barisic, Jácome Cunha, Ivan Ruchkin, Ana Moreira, João Araújo, et al.. Modelling Sustainability in Cyber-Physical Systems: A Systematic Mapping Study. 2022. ffhal-03616678f.

9. Gvozdev V.E., Bezhaeva O.Ya. The studies of strategies for ensuring the functional safety of hardware and software complexes based on system archetypes and architecture models. In: Proceedings of the International Conference on Electrotechnical Complexes and Systems, ICOECS (Ufa, Russia, November 2021). IEEE, 2021. P. 171–175.

10. Kovtunenko A., Bilyalov A., Valees S. Distributed streaming data processing in IoT systems using multi-agent software architecture // Internet of Things, Smart Spaces, and Next Generation Networks and Systems. Lecture Notes in Computer Science. 2018. Vol. 1118. P. 572–573.

11. Kovtunenko A., Timirov M., Bilyalov A. Multi-agent approach to computational resource allocation in edge computing // Internet of Things, Smart Spaces, and Next Generation Networks and Systems. Lecture Notes in Computer Science. 2019. Vol. 11660. P. 135–146.
12. Kovtunenko A., Klimova A. Agent-based distributed system for the realtime data-driven simulation of complex dynamic systems. In: Proceedings of the 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). Dublin, 2019.

Об авторах:
КОВТУНЕНКО Алексей Сергеевич, к.т.н., доцент, доцент кафедры информатики, ФГБОУ ВО «Уфимский университет науки и технологий», +7-960-380-30-10, askovtunenko@mail.ru.
ВАГАНОВА Дарья Евгеньевна, ассистент кафедры информатики, ФГБОУ ВО «Уфимский университет науки и технологий», +7-919-144-55-10, dfalshunova@mail.ru.
СЕРЕБРЯКОВ Павел Константинович, аспирант ФГБОУ ВО «Уфимский университет науки и технологий», +7-927-230-20-04, cspk@list.ru.

Metadata:
Title: Multi-Agent Approach of Software Design in Cyber-Physical Systems.
Author 1: Alexey Sergeevich Kovtunenko, Cand. Sci., Associate Professor of the Computer Science Department, Ufa University of Science and Technologies, 32 Zaki Validi st., Ufa, Bashkortostan, Russian Federation, askovtunenko@mail.ru, ORCID ID 0000-0002-3814-7310, Web of Science ResearcherID AAH-5198-2019, Scopus Author ID 57204176565.
Author 2: Darya Evgenyevna Vaganova, Assistant Pr5ofessor of the Computer Science Department, Ufa University of Science and Technologies, 32 Zaki Validi st., Ufa, Bashkortostan, Russian Federation, dfalshunova@mail.ru, ORCID ID 0009-0008-9828-3796.
Author 3: Pavel Konstantinovich Serebryakov, postgraduate student at Ufa University of Science and Technologies, 32 Zaki Validi st., Ufa, Bashkortostan, Russian Federation, cspk@list.ru.
Abstract: The paper considers the problem of software organization in cyber-physical systems (CPS). A concept is proposed according to which the main computing processes in CPS are receiving and processing streaming data. A mathematical model of distributed processing of streaming data in cyber-physical systems is proposed, which reflects the main requirements for CPS software: flexibility and survivability. The proposed model is implemented in the form of a software architecture that allows to significantly increase the efficiency of the software life cycle for CPS.
Keywords: cyber-physical systems, software architecture, streaming data, multi-agent systems, software agent.