УДК 004.896+629.58+001.891.57

10.25808/24094609.2019.30.4.002

ПРОГРАММНАЯ ПЛАТФОРМА ДЛЯ ПОСТРОЕНИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ УПРАВЛЕНИЯ МОРСКИХ РОБОТИЗИРОВАННЫХ КОМПЛЕКСОВ

Г.Д. Елисеенко, А.В. Инзарцев, А.М. Павин

Федеральное государственное бюджетное учреждение науки
Институт проблем морских технологий ДВО РАН[1]

Описывается программная платформа (ПП), используемая при проектировании систем управления для морских роботизированных комплексов (МРК), в том числе имеющих распределенную структуру. При разработке ПП учитывался опыт ИПМТ ДВО РАН в создании и эксплуатации широкого спектра средств подводной робототехники. Ключевой особенностью ПП является использование децентрализованного асинхронного событийно-ориентированного режима обмена специализированными пакетами данных (сообщениями) между всеми программными компонентами МРК. Помимо этого ПП предоставляет ряд средств, облегчающих проектирование, наладку и эксплуатацию систем управления (визуализация информационного трафика в системе, логирование и постобработка потоков данных, реконфигурирование программного обеспечения, симуляция работы отдельных компонентов МРК). Рассматривается «философия» построения ПП, обсуждаются особенности организации и функционирования ее компонентов.

ВВЕДЕНИЕ

Последние годы характеризуются все более массовым использованием группировок мобильных робототехнических систем, включающих разнородные типы роботов. Сказанное в полной мере относится и к морским роботизированным комплексам (МРК). В состав МРК могут входить один или несколько мобильных роботов, например телеуправляемый подводный аппарат, АНПА, безэкипажный катер, летающий беспилотный аппарат и т.д. Комплекс включает также пост управления, который может состоять из нескольких компьютеризированных рабочих мест (постов оператора). В свою очередь, каждый мобильный робот и каждое рабочее место оператора могут состоять из нескольких компьютеров, объединенных в локальную вычислительную сеть и имеющих разнородные (разноскоростные) соединения с другими элементами МРК (рис. 1).

Сложность разработки программного обеспечения для МРК заключается в необходимости интеграции множества разнородных модулей подводных роботов в едином информационном пространстве. Решение подобной задачи может быть упрощено за счет использования базовой программной платформы (ПП), предоставляющей набор типовых средств.

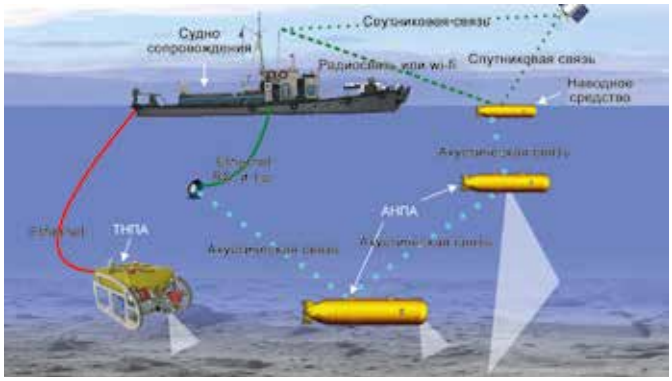

Рис. 1. Пример организации информационного взаимодействия элементов морских роботизированных комплексов

Можно выделить ряд основных функций, которые должна выполнять такая платформа:

1) поддержка высокоскоростного информационного обмена между программными компонентами внутри одного компьютера в режиме реального времени;

2) поддержка информационного обмена с минимальными накладными расходами между программными компонентами различных компьютеров,

[1] 690091, г. Владивосток, ул. Суханова, 5а. Тел.: +7 (423) 222-64-16. E-mail: eliseenko@marine.febras.ru, inzar@marine.febras.ru, pavin@bk.ru

находящихся в одной вычислительной сети и включающей разнородные физические каналы связи;

3) отображение существующих информационных потоков различных элементов комплекса в одном графическом интерфейсе оператора (например, отображение местоположения всех роботов и распознаваемых объектов на карте местности с привязкой к географическим координатам);

4) сбор и логирование (сохранение) всех или избранных информационных потоков внутри МРК, в том числе, в режиме «прореживания данных»;

5) постобработка и отображение сохраненных данных с возможностью воспроизведения («проигрывания») реальных ситуаций;

6) возможность симуляции (имитации) работы отдельных компонентов программного обеспечения, в том числе в ускоренном режиме и на основе сохраненной информации.

Ключевым моментом при разработке ПП является реализация первых двух из перечисленных выше пунктов. Известно несколько различных средств обеспечения информационного взаимодействия между процессами внутри операционной системы (ОС). Примером может служить использование общей памяти. Достоинством подхода является низкая ресурсоемкость. Однако подобный метод обладает рядом недостатков:

- невозможность обмена данными между процессами, находящимися на разных узлах ЛВС;
- необходимость разрешения конфликтов, связанных с одновременным доступом к одним и тем же областям памяти со стороны различных процессов;
- сложность организации событийно-ориентированной модели;
- наличие центрального звена (хранилища данных в оперативной памяти), что уменьшает надежность системы в целом.

Другим известным решением является применение клиент-серверной технологии [1], в том числе с использованием баз данных [2]. Такой подход обеспечивает синхронизацию обмена информацией между процессами, а также отсутствие их взаимоблокировок. К его недостаткам относятся:

- необходимость локализации сервера на одном из компьютеров сети, что влечет за собой все проблемы, связанные с наличием «центрального звена» (избыточное обращение к центральному узлу со стороны компонентов, информационное взаимодействие которых логически не связано с этим узлом);
- сложность выделения центрального звена в сети с изменяющейся топологией (например, при изменении конфигурации МРК во время работы);
- при использовании клиент-серверной технологии возникают временные задержки из-за необходимости поочередного доступа к хранилищу данных, что ограничивает область применения подобного информационного взаимодействия для систем реального времени;
- наличие единого хранилища данных снижает надежность системы в целом за счет увеличения риска потери всех логируемых данных.

Таким образом, анализ существующих решений позволяет выделить следующие требования к организации информационного взаимодействия компонентов МРК:

- необходимо применение распределенных (децентрализованных) событийно-ориентированных подходов при организации информационного взаимодействия;
- в основу информационного взаимодействия должны быть положены технологии с асинхронным режимом передачи данных (без установления соединения) с минимальными временными задержками при передаче информации для обеспечения режима реального времени.
- для обеспечения «горячего» реконфигурирования программные модули должны иметь возможность подключения к информационному обмену в системе в любой момент времени и восстановлению обмена после разрыва соединения;
- логирование данных должно быть распределенным без «паразитного» трафика по сети и не влиять на обмен сообщениями в случае выхода из строя бортового накопителя;
- компьютеры МРК должны иметь возможность функционирования под управлением различных операционных систем.

Далее в статье речь пойдет о специализированной программной платформе для МРК [3–5], созданной в ИПМТ ДВО РАН с учетом приведенных выше требований, а также опыта разработки и эксплуатации десятков образцов подводной робототехники различной конфигурации и назначения (в частности, при создании малогабаритного АНПА [6]).

Информационный обмен на базе распределенной программной платформы

Программная платформа реализована в виде кроссплатформенной библиотеки. Для построения информационного взаимодействия элементов МРК достаточно подключить библиотеку к каждой программе, участвующей в общем обмене данными. При этом не требуется установка каких-либо

дополнительных компонентов, утилит или постоянно действующих сторонних процессов системы. Библиотека отвечает за транспорт сообщений между компонентами МРК, организацию очереди сообщений, обработку конфигурационных файлов, логирование данных и поддерживает следующие основные механизмы обмена сообщениями:

- асинхронная посылка сообщений всем потребителям без блокирования поставщика;
- ожидание события прихода сообщений с организацией очереди фиксированного размера;
- доступ в любой момент времени к последнему пришедшему сообщению определенного типа без блокировки потребителя данных;
- подписка на публикацию, получение или считывание сообщений конкретного типа;
- регистрация таймеров в программе и ожидание событий срабатывания таймера.

В основу разработанной программной платформы положен децентрализованный принцип обмена сообщениями, когда каждый компонент системы является одновременно поставщиком и потребителем данных. Обмен между процессами осуществляется посредством сообщений, которые представляют собой особым образом структурированные блоки информации. Сообщения посылаются в асинхронном режиме. Такой подход позволяет избежать блокировки программ-поставщиков данных в случае нестабильной (или замедленной) работы программ-потребителей, а также в случае нестабильной связи между модулями. Кроме того, данное решение гарантирует функционирование модулей в режиме реального времени, насколько это позволяют ресурсы вычислительного комплекса. Все сообщения в ПП обладают уникальным идентификатором (именем), который равен символьному представлению типа данных. Такой подход гарантирует отсутствие пересечений идентификаторов у сообщений разных типов, так как имена структур в единой сборке программного обеспечения обязаны быть уникальными. Более подробную информацию о библиотеке обмена сообщениями можно найти в [4, 5].

Организация событийно-ориентированного подхода в системе

Событием в рассматриваемой системе может являться момент приема сообщения, срабатывание системного таймера или истечение таймаута, а также системные, внутренние и пользовательские события и прерывания, которые остаются вне области действия программной платформы. События, связанные с отправкой и получением данных, генерируются программами-поставщиками, а реакция на эти события обеспечивается программами-потребителями. При возникновении события, связанного с обменом, система производит следующую последовательность действий:

1. Процесс-источник данных публикует сообщение на указанный порт в соответствии с *ip*-адресами, определенными в настройках.
2. Сообщение передается в операционную систему и рассылается по всем указанным адресатам, включая (при необходимости) *localhost*-, *multicast*- и/или *broadcast*-вещания.
3. Принимаемое по сети или локально сообщение встает в очередь в операционной системе.
4. Процессы-потребители посредством библиотеки ПП проверяют порты (определенные именами сообщений), считывают сообщения из очереди и реагируют на них как на событие.

Формат сообщений программной платформы представлен на рис. 2. Заголовок сообщения имеет фиксированный размер и содержит минимальную служебную информацию для идентификации сообщения и доступа к другим разделам сообщения.

Раздел структуры передаваемых/принимаемых данных представляет собой последовательность байт определенного формата (специфичный формат пользователей, лежащий за рамками API библиотеки).

Рис. 2. Формат сообщений программной платформы

Служебный заголовок содержит информацию о времени публикации и размеры параметров служебной информации. В последнем разделе перечислены имена сообщения и хоста, с которого отправлено сообщение. Структура сообщения организована таким образом, что при наличии низкоскоростного канала связи можно пересылать не все сообщение, а только минимальный заголовок и данные (размер данных указан в заголовке). Таким образом, в каналах с ограниченной пропускной способностью будет потерян ряд служебной информации, но при этом размер сообщения сократится до размера переносимых данных и минимально-необходимого (для идентификации пакета) заголовка.

Деление информационных пакетов на команды, сообщения и данные на практике часто является условным. В рамках ПП считается, что команды указывают на необходимость некоторых действий со стороны компонентов системы и должны быть доставлены потребителям при наличии физического канала обмена, в то время как данные только участвуют в расчетах, поставляются периодически, и отдельные пакеты данных могут быть потеряны без серьезных последствий.

Организация межпроцессорного взаимодействия

Как говорилось выше, взаимодействие процессов между разными узлами ЛВС организовано путем обмена целостными пакетами данных – сообщениями. В отличие, например, от системы *ROS* [7], структура пользовательских данных внутри сообщений никак не регламентируется, полностью определяется разработчиками соответствующих программ и не требует никакой препроцессорной обработки. Особенностью реализованного в ПП асинхронного режима передачи сообщений является применение двух взаимодополняющих решений:

- повторная (дублирующаяся) отправка критически важных сообщений группе потребителей;
- использование нескольких сетевых карт и различных физических маршрутов между компонентами МРК.

Первое решение максимизирует вероятность получения критически важных команд группой потребителей. Такие команды являются однократными и посылаются достаточно редко. Использование для этих целей протоколов с гарантированной доставкой (например, *TCP*) приводит к необходимости поддержания соединений между модулями и использования специализированных идентификаторов программ, что в итоге накладывает ряд существенных ограничений при модернизации и масштабировании программного обеспечения (ПО) МРК. В то же время использование *TCP*-соединений также не гарантирует доставку в случае физического обрыва линии, но при этом приводит к необходимости создания дополнительных специфических обработок в реализации ПО.

Для обеспечения взаимодействия процессов, находящихся на разных компьютерах сети, выбран сетевой протокол *UDP*, который поддерживается большинством современных сетевых карт, операционных систем и компиляторов на уровне *POSIX*. В отличие от *TCP*, он менее ресурсоемок и специально предназначен для передачи пакетов данных в режиме реального времени. Недостатком *UDP* является отсутствие гарантии доставки пакетов. Однако при организации систем передачи данных в режиме реального времени в случае потери или искажения пакета можно выбрать только одно из двух:

- тратить ресурсы и время для гарантированной передачи пакета, который в итоге может оказаться не актуальным из-за временной задержки;
- отправить новый пакет данных и не задерживать очередь других сообщений.

Современные сетевые карты и коммутаторы (в особенности работающие в полнодуплексном режиме без коллизий) обеспечивают ничтожно малый процент потери пакетов. Потери случаются в основном при обрыве физической линии или переполнении буфера, в этом случае какими-то пакетами приходится жертвовать. При взаимодействии компонентов МРК в рамках одного компьютера опасность потери данных отсутствует. При распределенной работе (по сети) всегда есть возможность потери связи, вне зависимости от применяемого протокола.

В любом случае, при отсутствии необходимых для модуля данных необходимо особым образом обрабатывать подобные ситуации. Обработка видится проще, если сама программа не «заморожена» на приеме. Применяемый протокол *UDP* не требует создания подключений между поставщиками и потребителями данных. В случае с *TCP* для взаимодействия ряда программ, находящихся в тесном информационном обмене (например, если они образуют полносвязный информационный граф) пришлось бы создать множество подключений между каждой парой программ. У *UPD* имеется возможность отправки одного сообщения группе компьютеров и программ (*multicast*) или отправки широковещательного (*broadcast*) сообщения на определенную подсеть.

Для кардинального увеличения надежности взаимодействия программ, функционирующих в различных сегментах сети, в библиотеке межпроцессорного взаимодействия реализована возможность отправки данных по нескольким физическим маршрутам. Основной проблемой при реализации обоих перечисленных решений является не процесс отправки, а прием и фильтрация дублирующихся сообщений на стороне программы-потребителя. Для решения этой проблемы формат сообщений был дополнен циклическим идентификатором самого сообщения. Таким образом, реализованный в библиотеке алгоритм после приема сообщения по какому-либо из каналов связи сначала проверяет наличие данного сообщения в хранимой у себя истории.

Для уменьшения ресурсоемкости алгоритма в библиотеке хранится только циклический буфер идентификаторов (точнее, остаток от деления идентификатора на размер циклического буфера) ранее принятых сообщений, доступ к которому осуществляется со сложностью *O(1)*. Описанные решения позволили без ощутимого увеличения расхода вычислительных ресурсов реализовать режим доставки критически важных сообщений с использованием нескольких физических каналов связи.

Получение данных процессами-потребителями возможно одним из двух способов (рис. 3):

1. Просмотр последнего опубликованного сообщения определенного типа в любой момент времени. Такой подход имитирует обращение к серверу за последними по времени поступившими данными определенного типа. В действительности сервера не существует, а в локальной копии библиотеки потребителя всегда хранится актуальная информация обо всех сообщениях, на которые подписалась программа.

2. Ожидание сообщения определенного типа или одного из нескольких типов. В такой событийно-ориентированной модели программа-потребитель находится в ожидании пакета данных и заблокирована до тех пор, пока не придет сообщение или не истечет установленный тайм-аут. Этот подход позволяет легко организовать высокоскоростные циклы управления с минимальными временными задержками между событиями и выработкой реакции на них.

В случае когда необходимо взаимодействие группы программ в рамках одного компьютера, используется передача сообщений в обход сетевой карты компьютера. Такой режим необходим как для программ, изначально находящихся на одной ЭВМ, так и для локальной отладки программ, которые в дальнейшем будут интегрированы в общую информационную сеть. При передаче данных от одного процесса другому в рамках одного компьютера использован тот же

```cpp
#include <ipc.h>   // Подключаем библиотеку программной платформы.

// Объявляем пользовательские структуры данных.
struct Angles   {float yaw; float pitch; float roll;};  // Ориентация подводного робота.
struct Velocity {float vx; float vy;};                  // Скорость подводного робота.
struct Position {float x; float y;};                    // Рассчитанные координаты робота.

// Пример программы счисления пути, основанной на событийно-ориентированном подходе.
int main(int argc, char *argv[])
{
  ipc::Core             core(argc, argv);  // Создаем ядро обмена сообщениями.
  ipc::Timer            timer(core, 0.1);  // Создаем таймер и запускаем его с периодом 0.1 с.
  ipc::Receiver<Angles> angle(core, 10);   // Подписываемся на прием (с ожиданием) сообщений "Angles" (очередь до 10 шт).
  ipc::Reader<Velocity> velocity(core);    // Подписываемся на чтение (без блокировки) сообщений "Velocity".
  ipc::Sender<Position> position(core);    // Объявляем публикацию сообщений "Position".
  double last_time = core.time();          // Запоминаем текущее время.

  while (core.launched()) {                                  // Цикл ожидания приема сообщений и тиков таймера.
    if (angle.received()) {                                  // Наступило событие "прием сообщения Angles".
      double now_time = core.time();                         // Запоминаем текущее время.
      double dt = now_time - last_time;                      // Рассчитываем интервал времени между двумя событиями.
      last_time = now_time;                                  // Запоминаем новое время.
      velocity.read();                                       // Считываем текущую скорость без блокированию
      position._.x += dt * (velocity._.vy * sin(angle._.yaw)     // Рассчитываем новые
                          + velocity._.vx * cos(angle._.yaw));   // координаты робота
      position._.y += dt * (velocity._.vy * cos(angle._.yaw)     // с учетом ориентации по курсу
                          - velocity._.vx * sin(angle._.yaw));   // и текущей скорости.
      position.send();                                       // Публикуем новые координаты.
    }
    if(timer.received()) {/* ... */}                         // Выполняем некоторую работу по таймеру.
    if(core.timeout())   {/* ... */}                         // Выполняем некоторую работу по таймауту.
  }
  /* Здесь можно безопасно закрыть файлы, вызвать деструкторы и т.п. */
}
```

Рис. 3. Пример листинга программного кода с применением разработанной библиотеки

механизм, что и при передаче по сети. Единственным различием является то, что сообщения публикуются на локальный хост (*localhost*). При этом поддерживается весь функционал взаимодействия процессов, приведенный выше.

Использование единого механизма передачи данных (по сети и внутри одного компьютера) позволяет простым и прозрачным способом перенастроить программы на различные компьютеры или объединять модули в рамках одной ЭВМ при необходимости.

Причем в рамках одного компьютера гарантирована доставка сообщений без необходимости повторной отправки данных, а для разных сегментов сети применяется множественная отправка, в том чиле по разным физическим интерфейсам.

Реконфигурирование программного обеспечения и использование таймеров

В процессе создания ПО роботизированных комплексов часто возникает потребность в применении одних и тех же программ для различных элементов МРК. Например, программа спутниковой навигации может располагаться на судне обеспечения, на автономном и/или телеуправляемом подводном аппарате. Однако настройки таких программ часто бывают различны (меняются адрес устройств и инициализационные параметры). Кроме того, на этапе создания ПО настройки многих модулей могут быть неизвестны разработчикам.

Для обеспечения быстрой реконфигурации программных модулей, использующих программную платформу, разработана иерархическая конфигурационная модель, позволяющая без изменения исходного кода программ осуществлять их перенастройку в зависимости от решаемых задач. Для этого используется набор конфигурационных файлов, являющихся общими для всех модулей данной сборки ПО. Основной каталог конфигурационных файлов содержит базовый конфигурационный файл библиотеки *ipc.json*, который представляет собой общие настройки программной платформы. Если в каталоге имеется папка, соответствующая имени программы, то содержащиеся в ней настройки переопределяют примененные ранее (общие для всех программ). Это решение позволяет выделить как общую конфигурацию для группы модулей, так и переопределить специфичные настройки (исключения) для каждого модуля в отдельности. К базовым конфигурационным настройкам библиотеки ПП относятся:

1. Пул портов для прослушивания сообщений. Использование пула портов позволяет в рамках одного компьютера создавать группы программ, общающихся только между собой, но не имеющих общего обмена с другими программами на этом же компьютере.

2. Список *ip*-адресов потребителей данных. Наличие списка позволяет ограничить обмен рамками локального хоста, указать обмен только с конкретным компьютером в сети или со всеми компьютерами, находящимися в указанной *broadcast*-подсети или *multicast*-группе.

3. Параметр ускорения/замедления симуляционного (виртуального) времени, который предназначен для проведения ускоренного моделирования работы ПО и отладки подсистем аппарата (см. подробности ниже).

Кроме общих настроек программ в упомянутом каталоге могут располагаться файлы с инициализационными настройками специфичных (пользовательских) структур данных. Такой подход позволяет формализовать и структурировать специфичную конфигурацию всех модулей, а также избавляет исходный код программ от рутинных процедур считывания специфических (созданных разработчиком ПО) конфигурационных файлов. Все конфигурационные файлы представлены в формате *JSON* [8], который легко интерпретируется как человеком, так и машиной. Кроме того, для создания и редактирования файлов в пользовательском интерфейсе программной платформы применен стандарт *JSON-Schema* [9]. Следование данному стандарту позволяет динамически создавать графические элементы интерфейса, исключающие многие ошибки пользователя (например, ввод конфигурационного значения, явно больше или меньше возможного).

Одной из специфичных конфигураций программной платформы является параметр изменения хода системного времени. Этот параметр отличается от номинального значения (единицы) при проведении ускоренных или замедленных отладочных процедур. При этом один и тот же коэффициент изменения времени может одновременно применяться к группе избранных программ или к каждой программе в отдельности. Необходимость подобного масштабирования возникает при проведении модельных экспериментов, а также для оценки запаса производительности ЭВМ для разработанного ПО в целом. Для обеспечения унифицированного механизма работы с «системным временем» программная платформа содержит следующие средства:

- таймер, посылающий события (тики) фиксированного интервала вызвавшему его процессу (рис. 3, строка № 12 и 30). Таймер привязан к абсолютному

времени, что позволяет производить интегральные операции с данными (например, счисление пути) без опасности накопления интегральной ошибки за счет накапливающегося сдвига (убегания) таймера;

- таймаут ожидания сообщений (рис. 3, строка № 31). Позволяет при использовании событийно-ориентированной модели совершать обработку превышения таймаута некоторого события или обрабатывать факт отсутствия ожидаемых данных;
- относительную временную задержку. Приостанавливает выполнение программы на определенный интервал времени с высвобождением ресурсов ОС. Является аналогом функций *delay* и *sleep*, в которые добавлен функционал обработки сигналов досрочного завершения процесса (что позволяет корректно завершать программы по сигналам ОС и системы управления);
- расчет абсолютного текущего времени. Функционал аналогичен использованию структур *time_t* из стандарта *POSIX*, но имеет более удобный для проведения математических расчетов формат (время представлено в секундах с удвоенной точностью) и использует масштабный коэффициент изменения времени (см. выше).

Логирование пользовательских данных средствами программной платформы

В рамках ПП децентрализованный принцип обмена сообщениями был распространен на логирование данных и их отображение в графическом интерфейсе пользователя. В соответствии с принятым подходом каждый поставщик данных самостоятельно (посредством библиотеки) сохраняет публикуемые сообщения в локальное или удаленное хранилище, а также передает данные на компьютеры оператора для визуализации. Реализация этих функций находится в самой программной платформе и не имеет отдельного *API*. Такой подход гарантирует, что все публикуемые данные могут наблюдаться пользователем (разработчиком, системным администратором) и всегда сохраняются для дальнейшего анализа. При логировании данных каждое сообщение помещается в отдельный файл. Такой подход имеет ряд преимуществ перед часто используемым (например, в ROS) накоплением в единый файл:

- возможность загрузки относительно небольшого объема информации для быстрой оценки состояния отдельных подсистем МРК без необходимости скачивания всех накопленных данных;
- в случае повреждения какого-либо файла или носителя остальная информация остается нетронутой, что увеличивает надежность системы логирования данных в целом.

Файлы накопителя (рис. 4) представляют собой текстовые файлы в формате *JSON*, структура которых включает две основные секции: описание структуры логируемого сообщения и сами данные.

Описание структуры данных в самом файле накопления позволяет корректно интерпретировать находящиеся во второй секции данные в отсутствие какой-либо дополнительной информации.

Описание структур данных представляет собой *JSON*-схему и содержит исчерпывающую информацию о каждом сохраняемом параметре логируемого сообщения.

Следует отметить, что стандарт *JSON-schema* включает описание как отдельных полей, так и вложенных структур данных, что в итоге позволяет описать любые пользовательские структуры данных. К основному описанию, согласующемуся со стандартом *JSON-schema*, относятся:

- *type* – тип данных, который может быть объектом (структурой/классом в нотации C/C++), числом (в том числе перечисляемым типом и числом с плавающей запятой), строкой и булевым значением;
- *title* – наименование параметра (содержит текстовую строку) в графическом интерфейсе пользователя;

Рис. 4. Пример файла накопления данных

- *minimum/maximum* – минимальное и максимальное значение логируемого числового параметра;
- другие ключи, соответствующие стандарту [9].

Кроме того, при создании графического интерфейса пользователя, использующегося для просмотра накопленных данных, применяется ряд дополнительных ключей, которые не входят в стандарт *JSON-schema*:

- *unit* – текстовое описание единицы измерения параметра;
- *precision* – количество знаков после запятой, использующееся для логирования и отображения числовых величин в плавающем формате.

Децентрализованное логирование данных в разработанной программной платформе предназначено как для их последующего отображения, так и для проведения отладочных и симуляционных экспериментов. В частности, все сохраненные данные могут быть повторно «проиграны» одной из утилит программной платформы. С точки зрения программного обеспечения (использующего библиотеку ПП) информационный поток от реально действующих подсистем ничем не отличается от потока данных из накопителя. Для реализации данного функционала логирование данных производится по событию публикации их конкретным поставщиком, а не с постоянным интервалом времени. Принятый подход с разбиением всего потока логируемых данных на отдельные файлы-сообщения позволяет производить раздельное сохранение однотипных данных, опубликованных различными поставщиками. В свою очередь, применение меток времени (рис. 4 – поле *time* в ключе *storage*) позволяет в дальнейшем «проиграть» поведение всей системы, в том числе в режиме ускоренного или замедленного времени.

ЗАКЛЮЧЕНИЕ

Описываемая программная платформа реализована на языке программирования *C++*. В качестве средства реализации использовалась кроссплатформенная среда разработки приложений *Qt Creator* 5-й версии, однако исходный код основан на стандарте *POSIX* и не использует функционал библиотек *Qt*. Библиотека ПП собрана и протестирована в нескольких операционных системах (*QNX, Linux, MacOS, Windows, Android*) и используется в большинстве аппаратов, созданных в ИПМТ ДВО РАН. За годы эксплуатации принятые решения подтвердили свою высокую эффективность при выполнении задач, связанных с функционированием программного обеспечения в режиме реального времени на борту морских роботизированных комплексов. Дальнейшее развитие подхода видится в расширении функционала программной платформы применительно к групповому управлению подводными роботами.

Работа выполнена при поддержке Программы фундаментальных научных исследований по приоритетным направлениям, определяемым президиумом Российской академии наук, № 7 «Новые разработки в перспективных направлениях энергетики, механики и робототехники».

ЛИТЕРАТУРА

1. Reid S., Dale J. Inter-Process Communication. – URL: http://www.cs.cmu.edu/afs/cs/project/TCA/ftp/IPC_Manual.pdf (дата обращения: 11.11.2019).
2. An Overview of MOOS-IvP and a Users Guide to the IvP Helm. – URL: https://oceanai.mit.edu/ivpman/pmwiki/pmwiki.php (дата обращения: 11.11.2019).
3. Свидетельство о государственной регистрации программ для ЭВМ № 2019610890 РФ. Программная платформа для информационного взаимодействия функциональных компонентов в робототехнических системах / Елисеенко Г.Д., Павин А.М., Инзарцев А.В., Сидоренко А.В.; правообладатель ФГБУН ИПМТ ДВО РАН. – № 2018665497; заявл. 27.12.2018; опубл. 18.01.2019, Бюл. № 1.
4. Инзарцев А.В., Павин А.М., Елисеенко Г.Д, Родькин Д.Н., Сидоренко А.В., Лебедко О.А., Панин М.А. Реконфигурируемая кроссплатформенная среда моделирования поведения необитаемого подводного аппарата // Подводные исследования и робототехника. 2015. № 2 (20). С. 28–34.
5. Pavin A., Inzartsev A., Eliseenko G. Reconfigurable Distributed Software Platform for a Group of UUVs (Yet Another Robot Platform) // Proc. of the OCEANS 2016 MTS/IEEE. Monterey, USA, 2016.
6. Борейко А.А., Кушнерик А.А., Михайлов Д.Н., Павин А.М., Щербатюк А.Ф. Малогабаритный многофункциональный автономный необитаемый подводный аппарат «Х300» // Материалы VIII Всерос. науч.-техн. конф. «Технические проблемы освоения Мирового океана» (ТПОМО-8). Владивосток, 2019. С. 33–37.
7. Robot Operating System/ROS. – URL: https://www.ros.org (дата обращения: 11.11.2019).
8. Butler H., Daly M., Doyle A., Gillies S., Hagen S., Schaub T. The GeoJSON Format, RFC 7946. The Internet Engineering Task Force. – URL: https://tools.ietf.org/html/rfc7946 (дата обращения: 05.10.2019).
9. Understanding JSON Schema. – URL: https://json-schema.org/understanding-json-schema (дата обращения: 05.10.2019).