УДК 551.46.077:629.584

ДВУХУРОВНЕВОЕ ФОРМИРОВАНИЕ ЗАДАНИЙ ДЛЯ АВТОНОМНЫХ ПОДВОДНЫХ РОБОТОВ

Е.А. Голенков, Д.С. Одякова, Г.В. Тарасов, Д.И. Харитонов

Институт автоматики и процессов управления ДВО РАН[1]

Рассматривается подход к программированию автономных подводных аппаратов, позволяющий составлять задания вне зависимости от количества исполняющих задания аппаратов. В основе подхода лежит концепция двухуровневого программирования. На первом уровне программируется набор классов, поддерживающих объединение в различные комбинации. Классы первого уровня формируют декларативный язык описания задания, используемый на втором уровне. Синтаксис языка описания заданий отличается простотой и основан на XML. Задание представляется в виде дерева объектов. Интерпретатор задания используется только на стадии считывания задания. Исполнение задания происходит внутри классов, запрограммированных на первом уровне, что обеспечивает низкий уровень накладных расходов на интерпретацию.

ВВЕДЕНИЕ

Подводная робототехника является важнейшим инструментом освоения мирового океана. Множество проблем, решение которых десятилетия назад невозможно было представить без участия человека, в настоящее время успешно, а главное безопасно, решается автономными необитаемыми подводными аппаратами (АНПА) [2]. В качестве примера таких проблем можно назвать исследование дна, поиск затонувших объектов, спасательные работы, прокладку и ремонт подводных кабелей. В будущем можно ожидать, что автономные подводные роботы смогут заниматься ещё более сложными работами, например, подводным строительством, ремонтом судов без захода в порт, культивированием марикультур и т.д. Сложность таких работ состоит в том, что требуются согласованные действия не одного, а множества подводных аппаратов. При этом успешное выполнение работ зависит в первую очередь от корректности поставленного задания и способа разделения его между подводными аппаратами. Поиск подходящего языка и способа описания групповых заданий, снимающих ответственность с человека, ограниченного собственными возможностями, является одной из приоритетных проблем развития подводной робототехники.

Нами описан декларативный подход к программированию АНПА, который авторы реализовали при разработке нового языка описания заданий подводных роботов, используя опыт программированиямногопроцессорных вычислительных систем и работы над параллельными языками программирования. При этом основной задачей нового языка авторы видят не удобство описания заданий, а технологичность реализации, полагая, что этот язык может служить внутренним представлением заданий как на этапе программирования и подготовки задания, так и на этапе выполнения заданий при обмене информацией между автономными роботами и центрами управления.

1. Состояние дел с программированием подводных аппаратов

Практика разработки автономных необитаемых подводных аппаратов насчитывает уже несколько десятилетий. За это время были продемонстрированы возможности подводных аппаратов по получению подводных данных, включая измерение глубины моря, количества кислорода на разных глубинах и в разных временных интервалах. АНПА применялись при поисково-спасательных работах на этапе поиска вертолёта КА-27ПС, упавшего в море 26 марта 2003 года [12], а также в августе 2007 года в Северном Ледовитом океане на хребте Ломоносова экспедицией на атомном ледоколе «Россия», которой были выполнены исследования геологических характеристик дна на площади более 50 км^2 [11]. В последние годы усилия конструкторов направлены на уменьшение стои-

[1] 690041, Владивосток, ул. Радио, 5, тел./факс: 310-452, e-mail: cc@dvo.ru

мости и времени исполнения подводных операций, в частности за счёт разработки технологий группового выполнения заданий. Коллективное выполнение заданий представляет значительный практический интерес для таких заданий, как оперативный мониторинг водных акваторий и рельефа дна, поисковые и исследовательские работы, патрулирование и инспекция подводных объектов и сооружений [2]. В этом направлении известны разработки средств симуляции группового выполнения заданий, языков взаимодействия АНПА, языков управления группами АНПА.

Одной из основных проблем применения групп АНПА является формулирование групповых заданий. На ранних этапах постановки задач группового управления, в 1996 году, была предложена [8] модель программирования АНПА, состоящая из *стратегического*, *тактического* и *исполняемого* уровней. На *стратегическом* уровне задание описывается в форме набора правил на языке PROLOG, которые циклически обрабатываются дедуктивным интерпретатором, приводя к дискретной смене состояний системы по мере выполнения задания. На *тактическом* уровне реализуется набор функций, взаимодействующих, с одной стороны, с предикатами PROLOGa, возвращая значения TRUE или FALSE, а с другой стороны, с *исполняемым* уровнем посредством передачи асинхронных сообщений. На *исполняемом* уровне происходят взаимодействие с бортовыми подсистемами подводного аппарата и управление приборами, осуществляющими маневрирование и навигацию. Особенностью этой модели является самый верхний *стратегический* уровень, который обеспечивает необходимый пользователю уровень абстракции и удобства. В частности, в работе [9] продемонстрирована смена языка программирования PROLOG на сети Петри, которые позволяют визуализировать поток управления групповым заданием. Работа 2008 года [7] подтверждает полезность дополнительного уровня описания группового задания, в ней продемонстрирован графический подход к описанию групповых миссий, в котором задание формируется в виде отметок на карте, обозначающих регион задания, путевые точки, точки взаимодействия подводных аппаратов. Эти отметки используются для статического разделения подводного региона исследования между несколькими подводными аппаратами.

Задачи, решаемые на *тактическом* и *исполняемом* уровнях управления, практически совпадают с задачами управления одиночными подводными аппаратами. Длительное время развивающиеся языки управления одиночными подводными аппаратами, такие как AUVish [5], CCL [4], язык, разработанный в ИПМТ ДВО РАН [3], сформировали довольно полное представление о потребностях этих уровней. Базовая функциональность этих языков покрывает такие категории поведения АНПА, как маневрирование, навигация, обмен информацией, конфигурирование параметров, наблюдение и запись параметров оборудования, выполнение предопределённых функций.

Рассмотрим описание задания на примере языка, разработанного в ИПМТ ДВО РАН. Задание представляется в текстовой форме в формате XML [6]. Файл описания задания включает раздел конфигурации и план миссии. Раздел конфигурации состоит из метаданных, предусматривающих разделы координат маяков-ответчиков, координат запуска аппарата, географической привязки карт и ссылки на файл карты, ссылки на файл бортового журнала аппарата для записи реальной траектории движения. План миссии представляет собой императивную последовательность команд, транслируемую непосредственно в исполняемый код. Каждой поддерживаемой команде в XML-записи соответствует одноименный тег с атрибутами, соответствующими параметрам команды.

Множество команд управления АНПА объединено в библиотеку функций, состоящую из встроенных процедур и обращений к системным процессам. Набор процедур и процессов состоит из четырёх основных групп. К первой группе относятся процедуры, определяющие траекторию движения. Вторая группа включает системные процессы, служащие для организации работы бортовых устройств. Процессы третьей группы обеспечивают доступ к параметрам движения и данным сенсоров. К четвертой группе относятся процедуры, инициализирующие выполнение сложных алгоритмических программ нижнего уровня (алгоритмы отслеживания, обследования, поиска). Также имеются средства обработки сообщений, адресованных из системы управления. Они используются для обработки запросов от контрольно-аварийной системы аппарата и систем обнаружения.

2. Двухуровневый подход к формированию заданий

В настоящей статье для построения языка программирования групп АНПА была использована концепция многоуровневого программирования. По словам известного специалиста по языку Lisp Пола Грэма [1],

«Писать программы как набор уровней - это мощный метод, даже если он

используется в приложениях. При программировании "снизу вверх" программа пишется в виде серии уровней, каждый из которых служит языком для вышестоящего уровня. Чем большую часть своего приложения вы сможете превратить в язык для написания приложений подобного рода, тем удобнее будет повторно использовать ваш код».

Основным недостатком многоуровневого программирования является существенная потеря производительности при переходе с уровня на уровень с использованием интерпретатора языка следующего уровня или даже байт-кода. Чтобы сократить размер потерь на интерпретацию, авторами статьи предложен следующий подход к программированию подводных аппаратов:

- Фиксируется два уровня программирования.
- Первый уровень программируется на языке C++. При этом на следующий уровень передаётся информация только о классах, реализующих определённый тип функциональности.
- На втором уровне описывается задание для подводного аппарата в виде дерева объектов, классы которых определены на первом уровне. Связи в дереве объектов обозначают принадлежность нижележащих по дереву объектов к вышележащим.
- Связь между уровнями осуществляется при помощи специального объекта - модуля двухуровневого программирования, отвечающего за загрузку задания и формирование дерева объектов.

С точки зрения системного программиста подводного аппарата, модуль двухуровневого программирования представлен классом с простой функциональностью. Перед каждым выполнением задания объект доступа к модулю двухуровневого программирования должен быть проинициализирован. Далее задание для АНПА, являющееся текстовым файлом, либо читается из файла, либо получается по коммуникационным каналам. Выполнение задания происходит в синхронном режиме, то есть управление передаётся в модуль двухуровневого программирования и возвращается только по окончании задания. Исходный текст части программы, использующей модуль двухуровневого программирования, может быть следующим:

```
...
ExternalInterfaceLanguage PLmodule; // II уровень программирования AUV
while (GetTask(memBuffer))
{
PLmodule.Init(); // Подготовка в исполнению новой программы
PLmodule.ScriptFromMemory(memBuffer); для динамической программы
// либо PLmodule.ScriptFromFile(fileName); // Считывание из файла
PLmodule.RunScript(); // Запуск программы
}
...
```

3. Синтаксис языка описания заданий

Для языка описания задания был разработан примитивный абстрактный синтаксис, изображённый на рис. 1. Согласно этому абстрактному синтаксису задание состоит из списка объектов, каждый из которых может быть простым или сложным. Простой объект определяется *типом*, *именем* и *значением*. Причём *тип* и *значение* могут быть пропущены (на рис. 1 эти элементы подписаны в квадратных скобках). Сложный объект, кроме этого, имеет *параметры* в виде списка объектов, каждый из которых является либо простым, либо сложным. Далее сложный объект по отношению к своим параметрам будем называть собственником.

В качестве основы конкретного синтаксиса языка описания заданий использован стандарт XML. При этом простой объект описывается записью в форме, называемой пустым элементом. Возможны несколько вариантов записи простого объекта:

1. Полная запись простого объекта: **<Тип Name=»Имя», Value=»Значение»/>**
2. Пропущено *значение*: **<Тип Name=»Имя»/>**
3. Пропущен *тип*: **<Имя Value=»Значение»/>**
4. Пропущены *тип* и *значение*: **<Имя/>**

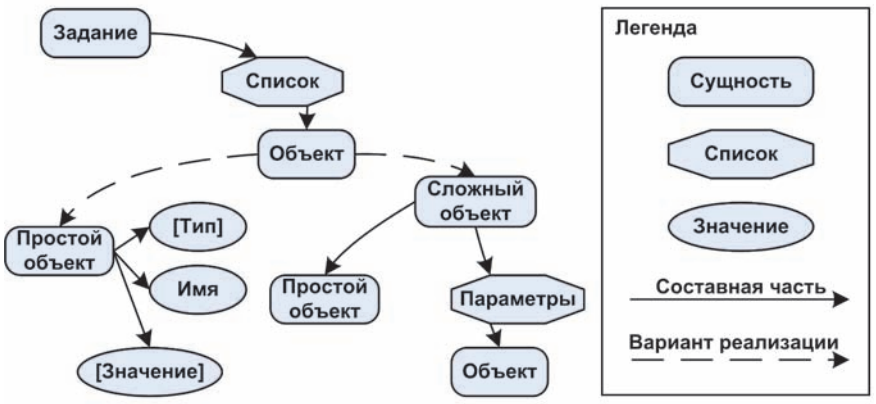

Рис. 1. Абстрактный синтаксис языка описания заданий

Запись сложного объекта состоит из открывающего тэга, содержимого и закрывающего тэга. Синтаксис открывающего тэга совпадает с синтаксисом простого объекта, за исключением предпоследнего символа «/». Содержимое сложного объекта является списком *параметров*. Синтаксис закрывающего тэга определяется правилами XML.

4. Пример задания

В качестве примера рассмотрим задание, в котором необходимо совершить обследование района дна, заданного географическими координатами, с целью поиска затонувшего объекта. При этом для обнаружения объекта или его фрагментов должны использоваться гидролокатор бокового обзора и электромагнитный искатель. Минимальный размер фрагментов задан равным 10 метрам. Для окончательной идентификации найденного объекта должна производиться фотосъёмка.

Ниже представлен текст описанного задания:

```
<Task Name="MissionOne">
    <SearchArea Name=»Volume1»>
        <Latitude1  Value=»42.33577»/>
        <Latitude2  Value=»42.30817»/>
        <Longitude1 Value=»131.17016»/>
        <Longitude2 Value=»131.23885»/>
        <GroundSearch Name=»Method»>
            <Height Value=»10»/>
            <Means>
                <GBO Name=»Src1»>
                    <GBO_Analyser Name=»Output»>
                        <MinDiameter Value=»10»/>
                        <MaxDiameter Value=»50»/>
                        <Output Value=»@Collector»/>
                    </GBO_Analyser>
                </GBO>
                <EMI Name=»Src2»>
                    <EMI_Analyser Name=»Output»>
                        <ExcessFactor Value=»30»/>
                        <Output Value=»@Collector»/>
                    </EMI_Analyser>
                </EMI>
            </Means>
        </GroundSearch>
    </SearchArea>
    <SetIntersection Name=»Collector»>
        <SourceCount Value=»2»/>
        <PathControl Name=»Method»>
            <Bypass Value=»360»/>
            <Distance Value=»5»/>
            <Means>
                <Camera Value=»Src»>
                    <Resolution Value=»UHigh»/>
                    <Freq Value=»High»/>
                </Camera>
            </Means>
        </PathControl>
    </SetIntersection>
</Task>
```

Исходный текст задания после синтаксического разбора может быть представлен в виде дерева, изображённого на рис. 2. Фигурами со скруглёнными углами на рисунке обозначены сложные объекты. Простые объекты, параметры сложных собраны в прямоугольники. Сплошными линиями на рисунке отображены синтаксические связи принадлежности между объектами, формирующие дерево объектов задания. Пунктирными линиями на рисунке отображены семантические связи - ссылки, о которых будет рассказано далее по тексту.

Представленное на рис. 2 дерево объектов задания отображает все взаимосвязи, существующие в задании, упрощая его прочтение. Исходный текст задания состоит из одного объекта *Task*, фиксирующего границы текста задания. Объект *Task* содержит объекты *SearchArea Volume1* и *SetIntersection Collector*, первый из которых описывает акваторию проведения исследовательских работ, заданную географическими координатами, а второй - условия проведения фотосъёмки. Объект *SearchArea Volume1* содержит единственный обязательный объект *GroundSearch Method*, тип которого назначает способ обследования акватории. Объекту *GroundSearch Method* назначен список *Means* средств обследования акватории, состоящий из объектов *GBO Src1* (гидролокатора бокового обзора) и *EMI Src2* (электромагнитного искателя), которым, в свою очередь, назначены параметры *GBO_Analyser Output* и *EMI_Analyser Output*, отвечающие за обнаружение аномальных по соответствующим характеристикам объектов на дне акватории. Объекты *GBO_Analyser Output* и *EMI_Analyser Output* передают координаты аномальных точек объекту

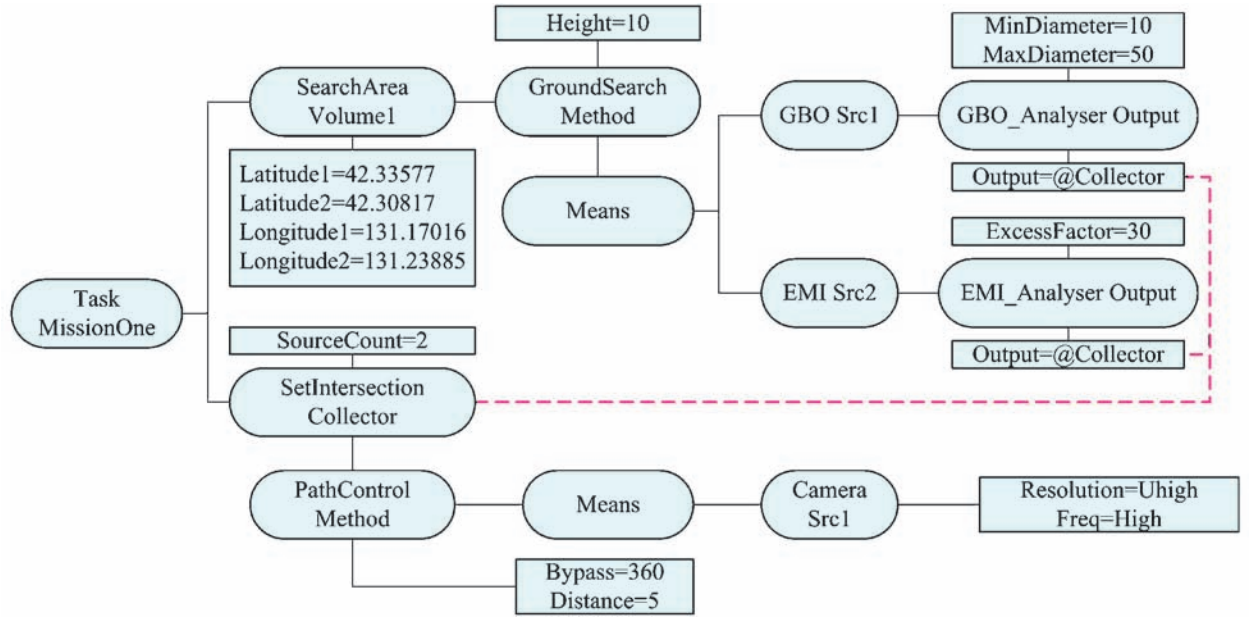

Рис. 2. Дерево объектов задания с отображёнными ссылками между объектами

SetIntersection Collector, формирующему новое множество координат, в которых присутствуют обе аномалии. Объект *PathControl Method*, определяющий траекторию обхода аномальных точек, является единственным обязательным параметром объекта *SetIntersection Collector*, ему назначен список *Means*, состоящий из единственного объекта *Camera Src1*, обязывающего применить камеру для фотографирования морского дна в точках с аномальными характеристиками.

5. Семантика языка описания заданий

Описание задания является декларативным, то есть порядок исполнения задания определяется не последовательностью объектов, а их взаимосвязями внутри задания. Каждый объект из списка объектов задания может начинать своё исполнение как только все его параметры будут определены. Семантика задания определяется классами объектов, запрограммированными на первом уровне, и взаимным расположением объектов в дереве синтаксического разбора задания. Как уже было сказано, при описании синтаксиса каждый объект определяется *типом*, *именем* и *значением*.

Имя объекта может иметь два значения, в зависимости от его собственника. Если собственник является списком, то *имя* объекта может быть произвольным и служит для адресации к объекту. Если собственник не является списком, то *имя* объекта имеет однозначное соответствие члену класса, запрограммированному на первом уровне. При этом членами класса могут быть указатели, что позволяет задавать объекту-параметру различный *тип*.

Тип объекта имеет взаимно однозначное соответствие с классами объектов, запрограммированными на первом уровне. Тип объекта может быть пропущен, если этому объекту сопоставляется член класса, запрограммированный на первом уровне, не являющийся указателем.

Значение объекта может быть в зависимости от типа объекта числом, текстовой строкой и ссылкой. Обработка всех численных значений производится в формате двойной точности, а затем при передаче значения объекту может производиться автоматическое приведение типа к одинарной точности или целому числу. Текстовая строка передаётся объекту без изменения, смысл и содержание текстового *значения* каждый класс, запрограммированный на первом уровне, определяет самостоятельно. Ссылка на объект записывается в форме $@name_1.name_2...name_n$, где символ @ нужен, чтобы на этапе синтаксического разбора и подготовки задания заменить ссылку на объект численным значением указателя. $name_1.name_2...name_n$ – это относительный адрес объекта, состоящий из последовательности имён $name_i$ объектов, являющихся параметрами объекта $name_{i-1}$. Поиск объекта с именем $name_1$ производится сначала в объекте-собственнике параметра, значение которого является ссылкой, затем в объекте-собственнике собственника и так далее, пока не будет найден объект $name_1$.

6. Разработка классов для описания задания

Управление АНПА условно делится на две подзадачи: во-первых, управление передвижением робота; во-вторых, управление различным бортовым оборудованием. Императивный поток команд управления одиночным АНПА формируется из декларативного задания, описывая траекторию передвижения аппарата и периоды подключения бортового оборудование на этой траектории. Для выполнения задания группой аппаратов оно должно быть разделено на несколько потоков команд управления одиночными АНПА. Разделение задания может быть статическим и динамическим. Статическое распределение заданий производится один раз до начала выполнения задания, поэтому является более простым и не требует от АНПА коммуникационных возможностей. Динамическое распределение заданий может осуществляться либо в режиме client-server, при этом АНПА получают очередную порцию задания от главного компьютера, либо в распределённом режиме, когда АНПА осуществляют самостоятельную координацию действий во время задания. В предложенном двухуровневом подходе к формированию заданий работа по разделению задания на несколько АНПА выполняется на нижнем уровне, что позволяет использовать один исходный текст задания для разных режимов выполнения, при этом для каждого режима выполнения задания реализуется собственный набор классов первого уровня.

Классы, запрограммированные на первом уровне, определяют прагматику языка описания. Двухуровневое программирование позволяет запрограммировать классы уже после того, как разработано задание, их использующее. Таким образом, выразительность языка описания заданий формируется в процессе составления заданий. А набор классов, необходимых для выполнения задания, может быть запрограммирован по-разному, в зависимости от условий выполнения задания.

Для описанного в разделе 4 примера задания на первом уровне должен быть запрограммирован следующий набор классов: *SearchArea, SetIntersection, GroundSearch, PathControl, GBO, GBO_Analyser, EMI, EMI_Analyser, Camera*. К моменту написания статьи этот набор запрограммирован для статического разделения задания между аппаратами.

На рис. 3 изображена диаграмма наследования классов, использующая для всех вышеуказанных классов четыре базовых: *AnArea, AMethod, AMean, ATool*. При этом базовый класс *AnArea* представляет функциональность, необходимую для описания территории исследования подводным аппаратом, класс *AMethod* - способ исследования территории, класс *AMean* - средства исследования, а класс *ATool* - дополнительные инструменты. Каждый базовый класс имеет простой интерфейс для взаимодействия с другими базовыми классами.

Интерфейс класса *AnArea*

```
bool InitPointCycle(); // подготовка к циклу по точкам
bool GetNextPoint(MarineCoords &X); // получение очередной точки
bool AddPoint(MarineCoords &X,int PointType); // для формирования региона извне
```

Интерфейс класса *AMean*

```
double GetDistanceRange(); // Дальность действия средства
bool TurnOn(MarineCoords &X); // Включить средство
bool TurnOff(MarineCoords &X); // Выключить средство
int TurnCount;
```

Интерфейс класса *AMethod*

```
double Init(int N); // Настроиться на подготовку N маршрутов
bool PrepareTrek(); // Подготовить N маршрутов роботов
bool UseMean(MarineCoords &X);
```

Интерфейс класса *ATool*

```
bool ProcessData(MarineCoords &X,int DataType,void *Data).
```

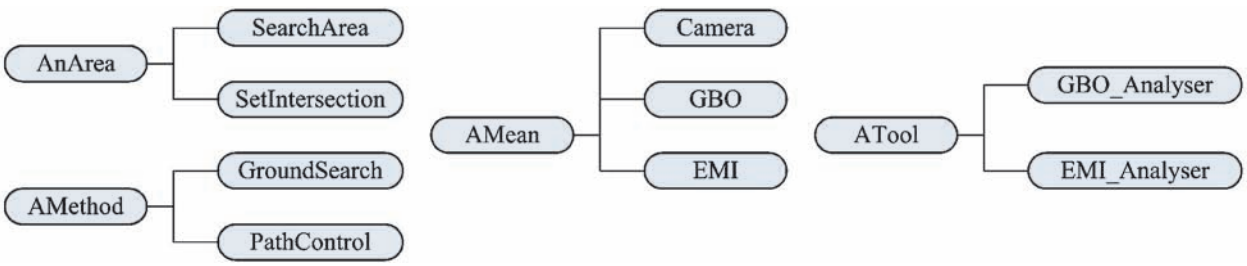

Рис. 3. Диаграмма наследования классов

Задача объекта класса *AnArea* - перечислить точки, через которые должен пройти подводный робот. В предложенном примере задания используются два класса (*SearchArea* и *SetIntersection*), выведенные из класса *AnArea*. Объект класса *SetIntersection* в примере задания принимает от объектов *GBO_Analyser* и *EMI_Analyser* координаты аномальных точек, составляя список только тех точек, в которых имеются обе аномалии. Объект класса *SearchArea* разобьёт участок с указанными географическими координатами на одинаковые квадраты и перечислит центры всех квадратов. При этом объект получит от средств исследования (*AMean*) информацию о дальности их действия (метод *GetDistanceRange*), чтобы рассчитать размер одного квадрата.

Объекты, выведенные из класса *AMethod*, отвечают за расчет маршрутов передвижения подводных аппаратов. Именно в этих объектах производится распараллеливание задания. Объект класса *GroundSearch* формирует обход множества точек на заданном расстоянии от дна. При этом для статического распараллеливания задания сначала рассчитывается траектория движения одного аппарата, затем эта траектория разделяется на равное количеству аппаратов число маршрутов. Разделение происходит путём последовательного отбора от полной траектории участка заданной длины. Объект класса PathControl формирует обход множества точек по часовой стрелке, причём вокруг каждой точки маршрута может происходить поворот подводного аппарата на заданный угол. Объект класса PathControl не распараллеливает задание, так как в реализованном статическом режиме разделения задания множество точек обхода до начала выполнения задания неизвестно. Распараллеливание обхода при динамическом режиме разделения задания является оптимизационной задачей и может производиться, например, с применением генетических алгоритмов [10].

Полученные объектами класса *AMethod* маршруты транслируются в текст программы управления аппаратом. При этом на подходе к точкам маршрута вызываются интерфейсные функции *TurnOn* объектов класса *AMean*, которые увеличивают счётчик TurnCount, а при первом увеличении счётчика также дополняют программу управления инструкциями включения соответствующих бортовых приборов. При выходе из зоны действия точки маршрута вызываются интерфейсные функции TurnOff, уменьшающие счётчик TurnCount и добавляющие команды выключения приборов при обнулении счётчика. Объекты, выведенные из класса AMean, передают данные обработчикам, выведенным из класса ATool, позволяющим организовать сложные взаимодействия между объектами.

7. Алгоритм загрузки задания

Алгоритм загрузки задания получает данные либо из файла, либо в виде указателя на буфер памяти, содержащий исходный текст. Задачей алгоритма является построение из текста задания дерева объектов задания, аналогичного изображённому на рис. 2. Текстовая форма XML позволяет записывать дерево объектов задания в соответствии с правилом обхода сверху вниз, справа налево. При восстановлении дерева выполняется следующая последовательность шагов:

1. Сначала строится дерево синтаксического разбора описания задания, являющееся шаблоном для построения дерева объектов задания. В каждом узле дерева заполняются атрибуты <тип, имя и значение>.

2. Каждому узлу ставится в соответствие прототип - описание класса, запрограммированного на первом уровне. Если в прототипе узла собственника существует параметр с таким же именем, то узлу ставится в соответствие прототип параметра. Тип каждого узла проверяется на совместимость с прототипом собственного параметра или прототипом узла собственника.

3. Восстанавливается структура задания. Для этого производится обход узлов дерева разбора от корня к листьям. Узлам, которым сопоставлены прототипы параметров, добавляются ссылки на члены классов. Для узлов, которым не сопоставлены прототипы параметров, заводятся объекты соответствующих классов, эти объекты регистрируются у собственников.

4. Производится инициализация задания. Для этого производится обход узлов дерева от корня к листьям и обратно. При движении от корня к листьям объектам передаются их значения, записанные в тексте задания. При движении от листьев к корню вызывается функция инициализации объекта, к этому моменту все внутренние параметры уже являются проинициализированными, что позволяет производить корректную настройку алгоритмов, описанных в тексте задания.

ЗАКЛЮЧЕНИЕ

Мы рассмотрели подход к программированию заданий автономных подводных роботов, который позволяет описывать задания в декларативном виде. Благодаря отходу от императивной парадигмы описания задания это позволяет описывать как одиночные, так и групповые миссии. Программирование на двух уровнях позволяет, во-первых, формировать семантику языка описания заданий в соответствии с текущими потребностями, а во-вторых, адаптировать программную среду для конкретных условий выполнения задания. Разобран пример, демонстрирующий текст и способ описания заданий, а также представлена реализация набора классов, используемых в примере, осуществляющая статическое разделение задания между группой подводных роботов. Запрограммированный набор классов предназначен не для реального управления роботами, а исключительно для демонстрации возможностей описанного подхода. Дальнейшая работа над языком описания заданий предусматривает разработку реализации классов для динамического разделения заданий и выполнения задания непосредственно на подводных роботах. Следует также отметить, что использование XML в качестве основы синтаксиса языка описания заданий предусматривает простую автоматизацию построения заданий, то есть для удобства пользователя может быть запрограммирован графический интерфейс, формирующий задания в текстовом виде.

Суммируя вышеизложенное, можно сказать, что получен механизм двухуровневого программирования заданий автономных подводных роботов, разработан простой алгоритм считывания задания, накладные расходы на интерпретацию задания ложатся только на этап построения дерева объектов, что обеспечивает быстрое выполнение всего задания.

Работа выполнена по Программе фундаментальных исследований Президиума РАН №1.

ЛИТЕРАТУРА

1. Грэм П. Языки программирования через сто лет [Электронный ресурс] // Компьютерра-Онлайн. 2004. URL: http://www.computerra.ru/hitech/35042/.

2. Агеев М.Д., Киселев Л.В., Матвиенко Ю.В. и др. Автономные подводные роботы. Системы и технологии // Под общ. ред. М.Д.Агеева. М.: Наука, 2005. 398с.

3. Инзарцев А.В., Матвиенко В.Ю. Визуальная среда разработки заданий для автономных подводных роботов // Подводные исследования и робототехника. 2008. №1. С. 5-10.

4. Duarte C., Martel G., Buzzel C. et al. A Common Control Language to Support Multiple Cooperating AUVs // Proc. of the 14th Int. Symp. on Unmanned Untethered Submersible Technology (UUST05), New Hampshire, August 21-24, 2005. New Hampshire, 2005.

5. Rajala A., O'Rourke M., Edwards D.B. AUVish: An Application-Based Language for Cooperating AUVs // Proc. of OCEANS 2006, Boston, MA, September 18-21, 2006. Boston, 2006.

6. Extensible Markup Language (XML) 1.0 (Fifth Edition) // World Wide Web Consortium (W3C). 2008. URL: http://www.w3.org/TR/xml/.

7. Giger G., Kandemir M., Dzielski J. Graphical Mission Specification and Partitioning for Unmanned Underwater Vehicles // Journal of Software (JSW). 2008. Vol. 3, No. 7. P. 42-54.

8. Healey A.J., Marco D.B., McGhee R.B. Autonomous Underwater Vehicle Control Coordination Using A Tri-Level Hybrid SoftwareArchitecture // Proc. of IEEE Int. Conf. on Robotics and Automation, Minniapolis, MI, April 22-28, 1996. P. 2149-2159. Minniapolis, 1996.

9. Healey A.J., Marco D.B., Oliveira P. et al. Strategic Level Mission Control - An Evaluation of CORAL and PROLOG Implementation for Mission Control Specifcations // Proc. of 6th Int. Work. on Underwater Robotics Program (IARP'96), Toulon, France, 2-6 June 1996. Toulon, 1996.

10. Киселев Л.В., Инзарцев А.В., Бычков И.В., Максимкин Н.Н., Хмельнов А.Е., Кензин М.Ю. Ситуационное управление группировкой автономных подводных роботов на основе генетических алгоритмов // Подводные исследования и робототехника. 2009. №2(8). С. 34-43.

11. Инзарцев А.В., Каморный А.В., Львов О.Ю., Матвиенко Ю.В., Рылов Н.И. Применение автономного необитаемого подводного аппарата для научных исследований в Арктике // Подводные исследования и робототехника. 2007. №2(4). С. 5-14.

12. Поиск и подъём вертолета Ка-27ПС. Март-апрель 2003 года. Тихоокеанский флот // Морской сборник. 2004. №5. С. 55-63.