УДК 004.624

РЕШЕНИЕ ЗАДАЧИ СОЗДАНИЯ РЕКОНФИГУРИРУЕМОГО ИМПОРТА ВХОДНЫХ ДАННЫХ ДЛЯ ПРОГРАММНОЙ СИСТЕМЫ ВОСПРОИЗВЕДЕНИЯ ПАРАМЕТРИЧЕСКОЙ ПОЛЕТНОЙ ИНФОРМАЦИИ

С.В. Федченко

Институт компьютерных технологий
Национального авиационного университета,
проспект Космонавта Комарова, 1.
E-mail: sergey@gp-opi.com.ua

Рассмотрен ряд недостатков большинства существующих систем воспроизведения параметрической информации. Определено решение задачи создания реконфигурируемого импорта входных данных для программной системы воспроизведения параметрической полетной информации в рамках требований к системам CNS/ATM.

A number of shortcomings of the most existing parametrical data playback systems is analysed. Solution is formulated for configurable import of input data design for flight data playback programme system in the context of CNS/ATM system requirements.

Введение

В настоящее время существует широкий спектр задач, в решении которых одним из этапов является процесс воспроизведения ППИ. Суть такого процесса заключается в предоставлении оператору определенной системы различной параметрической информации, которая лежит в основе этой системы. Под понятием параметрической информации в данном контексте подразумеваются последовательности формируемых рассматриваемой системой числовых величин образующих развернутые во времени параметры. С точки зрения объема предоставляемых функциональных возможностей наиболее целесообразным для ее воспроизведения на сегодняшний день является использование цифровой вычислительной техники со специальным программным обеспечением. В этом случае формирование и структуризация параметров осуществляется аппаратной частью, представляемой различными датчиками, преобразователями сигналов, устройствами регистрации, интерфейсами и т.п. Программная часть системы выполняет прием параметров непосредственно в память компьютера, их обработку, формирование новых параметров на базе полученных, сохранение их в требуемом формате, собственно воспроизведение, а также другие функции.

Типичным примером такой системы может быть система воспроизведения полетной информации выполняющая визуализацию траекторий полета воздушных суден (ВС). Исходя из требований глобального плана перехода на новую организацию воздушного движения (ВД), каждое ВС должно автоматически передавать данные о своем местоположении, курсе, скорости и прочую полезную информацию выдаваемую системой управления полетом (FMS) а также принимать и обрабатывать аналогичную информацию других ВС [1]. Ввиду широкого разнообразия типов ВС и их конструктивного различия такая информация не может на начальном этапе обрабатываться единым образом, что в свою очередь связано с рядом проблем поддержки каждого конкретного ВС при управлении ВД, а введение различных систем будет не эффективным и экономически не целесообразным. Наиболее оптимальным подходом в данном случае будет создание универсальной реконфигурируемой программной системы способной на начальном этапе обрабатывать информацию с различных типов ВС и сводить ее к унифицированному виду для дальнейшей обработки автоматизированными системами [2].

Постановка задачи

Как правило, программные части систем ППИ со сложной структурой данных разрабатываются специально для каждой конкретной системы. В целях оптимизации процессов обработки протоколов обмена с аппаратными устройствами создаются программные структуры с минимальным количеством описателей входной информации. Таким образом, каждая специализированная программная система оказывается наиболее подходящей для заданных условий своего применения, а все ее функциональные блоки разрабатываются с максимальным соответствием друг другу (рис. 1). Однако, на практике, как в процессе сопровождения программной части, так и в процессе развития обслуживаемой системы очень часто возникает необходимость

модификации уже существующих программных средств. При этом первичное предназначение продолжает оставаться актуальным. Аналогичная необходимость возникает также и при появлении похожих по функциональным характеристикам задач с участием других контролируемых систем. В ряде случаев для достижения новой функциональности достаточно введения в существующих средствах незначительных изменений путем рефакторинга [3] и обеспечение контроля версий программного продукта (рис. 2).

В других случаях требуются более кардинальные изменения отдельных элементов вплоть до полного отказа от варианта их использования. Введение новых элементов связано с наращиванием перечня описателей входной информации, что в свою очередь приводит к громоздкости описательных структур и усложнению системы в целом. Решением данной проблемы может стать создание альтернативных описательных структур и обработчиков исключений (рис. 3). Однако такое решение нарушает первичную архитектуру и значительно усложняет сопровождение программной системы. В обоих случаях имеют место нерациональные трудозатраты, а также неэффективное использование ресурсов вычислительной техники.

Рис. 1. Структура программной системы с минимальным количеством описателей входной информации (оптимальная архитектура)

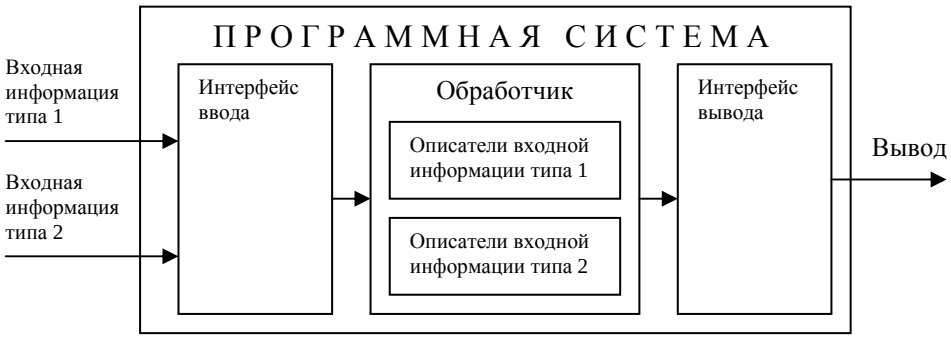

Рис. 2. Структура программной системы с расширенным количеством описателей входной информации (незначительные изменения при модернизации)

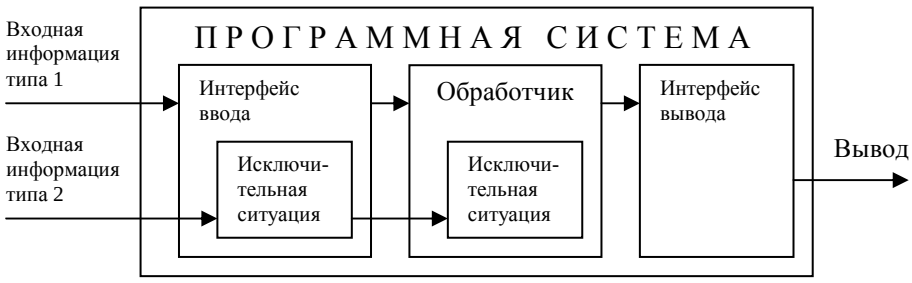

Рис. 3. Стуктура программной системы с обработчиком исключительной ситуации (существенные изменения при модернизации)

Практически данные для описательных структур обычно оформляются в виде внешних файлов, используя которые, программная система получает возможность извлекать конкретные значения параметров из общего потока информации. Данный подход позволяет избежать дублирование кода обработчика для однотипных параметров, однако, при их качественном изменении разработчики вынуждены добавлять новый обработчик или вводить исключительную ситуацию. В таблице 1 приведены описатели параметров, используемые эксплуатируемыми в настоящее время комплексами обработки полетной информации [4].

Таблица 1. Описатели параметров системы регистрации полетной информации самолета Boeing767 в програмных комплексах "Славутич" (MONSTR) и Avionica

Регистрируемый параметр	"Славутич" (MONSTR)	Avionica
Вертикальная перегрузка	N NY Верт. перегрузка 6 4095 40 10 220 5 0 8 4 12 20 28 36 44 52 60 -2 0 0 0 3 0.00228938 -3.375 4096	4,0xf,0,0xfff,0.00228938,-3.375000,ACCver,g ,lin_func_2,2,~ 12,0xf,0,0xfff,0.00228938,-3.375000,ACCver_2,g ,lin_func_2,2,~ 20,0xf,0,0xfff,0.00228938,-3.375000,ACCver_3,g ,lin_func_2,2,~ 28,0xf,0,0xfff,0.00228938,-3.375000,ACCver_4,g ,lin_func_2,2,~ 36,0xf,0,0xfff,0.00228938,-3.375000,ACCver_5,g ,lin_func_2,2,~ 44,0xf,0,0xfff,0.00228938,-3.375000,ACCver_6,g ,lin_func_2,2,~ 52,0xf,0,0xfff,0.00228938,-3.375000,ACCver_7,g ,lin_func_2,2,~ 60,0xf,0,0xfff,0.00228938,-3.375000,ACCver_8,g ,lin_func_2,2,~
Пролет дальнего, ближнего и среднего маркерных маяков	N IOM OM ILS 140 2 -32 15 0 0 0 2 64 64 6 1 N IMM MM ILS 140 1024 -32 110 0 0 2 58 58 6 2 N IIM IM ILS 140 512 -32 10 0 0 0 2 58 58 6 4 N MM Пролет маркерного маяка 140 1536 1114 2 0 1 4 58 64 58 64 6 8	58,0xf,10,0x1,1.000000,0.0000,MIDDLE_MARKER, ,lin_func_2,0,1-NO MARKER 0-MARKER 58,0xf,9,0x1,1.000000,0.0000,INNER_MARKER, ,lin_func_2,0,1-NO MARKER 0-MARKER 64,0xf,1,0x1,1.00000,0.0000,OUTER_MARKER, ,lin_func_2,0,1-NO MARKER B 0-MARKER

Главным требованием в обеспечении возможности осуществлять нетрудоемкую модернизацию существующей программной системы является анализ и разработка соответствующей универсальной реконфигурируемой архитектуры. При этом показатели ресурсообеспечения функционирования и затраты на разработку такой системы не должны превышать аналогичные показатели специализированных систем, а затраты на сопровождение и модернизацию должны быть минимальными.

Также в программной системе должна быть предусмотрена возможность ее модернизации сторонними разработчиками на основании договоров и соглашений всех участвующих сторон. Средства разработки (SDK) в таком случае могут быть представлены поставляемыми исходным текстом, документацией и другими файлами.

Таким образом, целью работы является изложение материала относительно проектирования реконфигурируемой программной системы воспроизведения ППИ согласно выдвинутым требованиям в соответствии с ранее сформулированной концепцией [5].

Реализация альтернативной архитектуры

Формируемая устройствами оцифрованная информация представляет собой цифровые значения различной битности регистрируемые с различной частотой. Как правило, эта информация при сохранении или передаче группируется в кадры фиксированной частоты следования и содержания. Однако, возможны также и случаи с изменяемой частотой регистрации параметров и нефиксированными по размеру кадрами, например, запись информации в виде оцифрованного аналогового сигнала или передача информации в виде пакетов данных. Таким образом, в общем случае описание параметра не должно включать в себя временные и структурные характеристики, а каждому имеющемуся значению параметра должно соответствовать условное время его регистрации (позиция) (рис. 4).

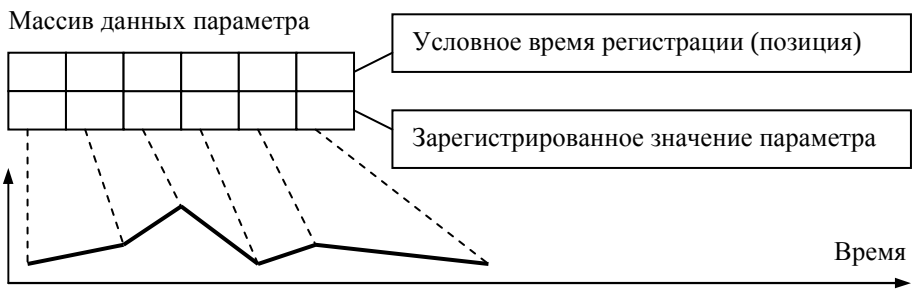

Рис. 4. Представление регистрируемых значений параметра

Максимальное количество значений разделенных условным временем регистрации для одного параметра определяется величиной битности условного времени регистрации, а общая максимальная продолжительность регистрации может быть рассчитана по формуле:

$$T = t * 2^n,$$

где t – промежуток времени между двумя ближайшими условными временами регистрации, n – битности условного времени регистрации. Например, для параметра, регистрируемого с частотой 1 Гц и описываемого позицией 64 бита, общая максимальная продолжительность регистрации будет составлять:

T = 1 * 2^64 = 18446744073709551616 секунд, что вполне приемлемо даже для систем с высокочастотной регистрацией параметров.

Сама регистрируемая величина в виду широкого разнообразия возможных значений и в целях экономии памяти компьютера должна задаваться индивидуально для каждого параметра. Изначально может быть описан следующий, наращиваемый в процессе модернизации программного обеспечения перечень значений:

```
enum PARAMETERVALUETYPE
{
        PVT_BOOL,
        PVT_BYTE,
        PVT_CHAR,
        PVT_WORD,
        PVT_SHORT,
        PVT_DWORD,
        PVT_INT,
        PVT_UINT64,
        PVT_INT64,
        PVT_FLOAT,
        PVT_DOUBLE,
        PVT_STRING
};
```

В соответствии с концепцией [4] программная система разделяется на основную универсальную часть, выполняющую визуализацию и подключаемые модули преобразования информации для предоставления данных в основную часть. Согласно такому принципу основной частью служит приложение Windows, выполняющее воспроизведение получаемых данных и другие пользовательские функции. А подключаемые модули могут быть представлены динамически подключаемыми библиотеками (dll) либо исполняемыми модулями, имеющими различную внутреннюю и внешнюю конструкцию и перечень стандартных экспортируемых атрибутов и функций, включающих две не несущие функциональной нагрузки строки названия (g_szPluginName) и описания (g_szPluginDescription) этого модуля, опциональные пользовательские функции просмотра и редактирования его настроек (Properties), загрузки (Load) и сохранения (Save) во внешний файл (szPath), инициализации (Init) и финализации (Delete), а также обязательные функции получения общего количества параметров (GetParametersNum), каждого из них в отдельности (GetParameter), и функция передачи данных параметра (GetParameterValues):

```
#ifdef MVDLL_EXPORTS
#define MVDLL_API __declspec(dllexport)
#else
#define MVDLL_API __declspec(dllimport)
#endif

extern MVDLL_API TCHAR *g_szPluginName;
extern MVDLL_API TCHAR *g_szPluginDescription;

MVDLL_API void Properties();

MVDLL_API HRESULT Load(TCHAR *szPath);
MVDLL_API HRESULT Save(TCHAR *szPath);
MVDLL_API HRESULT Init();
MVDLL_API HRESULT Delete();

MVDLL_API DWORD GetParametersNum();
MVDLL_API HRESULT GetParameter(DWORD dwParameter, PARAMETER *pparameter);
MVDLL_API HRESULT GetParameterValues(DWORD dwParameter, void *pValues, __int64 *pi64Positions, int
*piValuesCount);
```

Посредством функции GetParameter основная программа получает содержащиеся в подключаемом модуле описания параметров, для каждого из которых могут задаваться используемые отображающей системой

короткое (tag) и длинное (name) имена, и единица измерения (measure), представляемые строками неограниченной длинны. Остальные описатели указываются в виде строкового массива в формате:

"<имя><пробел><описание>", где имя непосредственно определяет название описателя. Данный подход позволит задавать любые строковые описания параметра и в процессе развития системы без необходимости рефакторинга добавлять для них соответствующие обработчики. Так, например, может быть указан диапазон изменения величины параметра: "Range 0 – 360".

Для индикации в описании параметра также может быть указано сбойное значение, задаваемое для удобства сравнения в виде аналогичном другим значениям этого параметра. Таким образом, отображающая система сможет корректно отмечать условными обозначениями или пропусками места сбойной регистрации или передачи. Сама же передача осуществляется посредством функции GetParameterValues, аргументами которой выступают порядковый номер запрашиваемого параметра, указатели на буфер значений и буфер позиций, а также указатель на количество запрашиваемых значений. Причем в первом элементе буфера массива позиций устанавливается стартовая позиция запроса, а количество запрашиваемых значений может быть отрицательным, в случае если системе требуется получить данные в обратном порядке от стартовой позиции. Такой подход позволяет свести до минимума количество передаваемых аргументов и производить всю передачу данных одной функцией. Для удобства получения значений в класс параметра включены все необходимые атрибуты:

```
class CParameter
{
public:
	DWORD         m_dwNumber; // Порядковый номер

	TCHAR           *m_szTag; // Короткое имя
	TCHAR          *m_szName; // Название
	TCHAR       *m_szMeasure; // Единица измерения
	DWORD        m_dwDataNum; // Количество описателей
	TCHAR        **m_pszData; // Описатели

	PARAMETERVALUETYPE m_pvt; // Тип значений

	void        *m_pBabValue; // Сбойное значение

	DWORD     m_dwBufferSize; // Размер буффера значений
	void          *m_pValues; // Значения
	__int64 *m_pi64Positions; // Позиции
	int       m_iValuesCount; // Текущее количество значений

	CParameter() { ZeroMemory(this, sizeof(CParameter)); }
	~CParameter()
	{
		delete [] m_szTag;
		m_szTag = NULL;
		delete [] m_szName;
		m_szName = NULL;

		delete [] m_szMeasure;
		m_szMeasure = NULL;

		if (m_pszData)
		{
			for (DWORD dw = 0; dw < m_dwDataNum; dw++) delete [] m_pszData[dw];
			delete [] m_pszData;
			m_pszData = NULL;
		}
		m_dwDataNum = 0;

		FreeValuesBuffer(m_pvt, m_pBabValue);

		m_dwBufferSize = 0;
		m_iValuesCount = 0;
		delete [] m_pValues;
		m_pValues = NULL;
		delete [] m_pi64Positions;
		m_pi64Positions = NULL;
	}
};
typedef CParameter PARAMETER, *LPPARAMETER;
```

Управление работой подключаемого модуля осуществляется с помощью внутреннего класса CLink основной программы, который содержит полученные описания параметров и отображающие элементы (рис. 5). Создаваемый объект этого класса может инициализироваться содержимым своего внешнего файла описателя либо конфигурируется пользователем вручную. При этом производится подключение заданного модуля, импортируются его атрибуты и функции. Инициализация завершается вызовом функций Load и Init подключаемого модуля (если они были успешно импортированы). В процессе работы объекта соединения основная программа запрашивает у подключаемого модуля значения параметров, отображаемых на экране с помощью текстовых надписей, графиков, трехмерных траекторий и объектов (рис. 6).

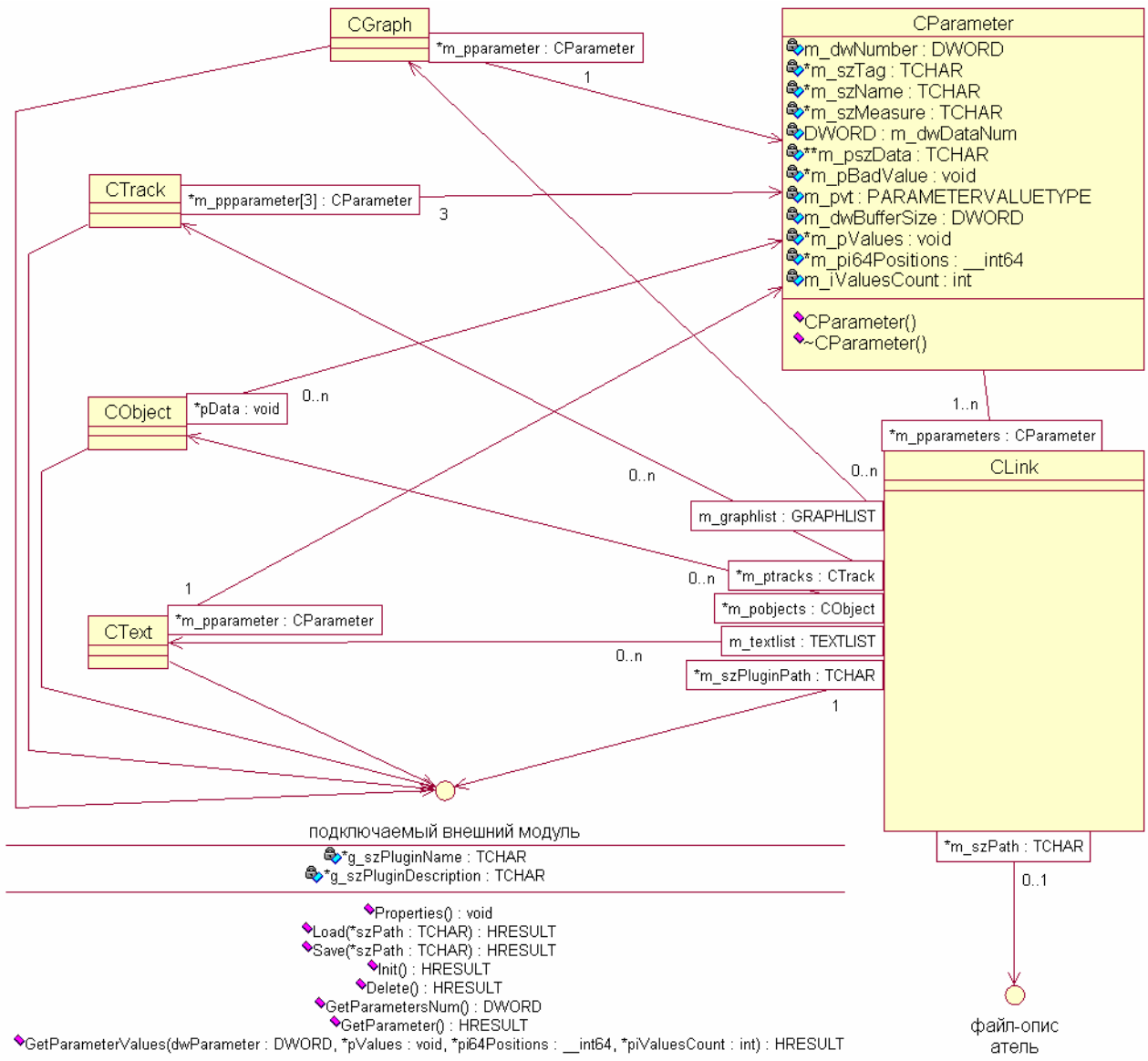

Рис. 5. Rational-диаграмма классов основной программы

Причем запрос происходит всегда только по активной зоне значений параметра и в зависимости от текущего положения курсора, в связи с чем в подключаемом модуле для ускорения произвольного доступа к значениям параметра может быть предусмотрена система кэширования данных. Например, для графиков запрашиваются только значения, отображаемые в текущий момент времени на экране и запрос происходит от текущего положения курсора, как в положительном, так и в отрицательном направлениях, что позволяет быстро восстанавливать состояние основной программы при ее закрытии и повторном открытии в процессе воспроизведения информации в режиме реального времени. Для объектов и надписей запрашиваются текущие значения, соответствующие позиции курсора, для траекторий – все допустимые значения. Работа объекта соединения завершается по команде пользователя или автоматически при закрытии основной программы. При этом вызываются функции Save и Delete подключаемого модуля (если они были успешно импортированы).

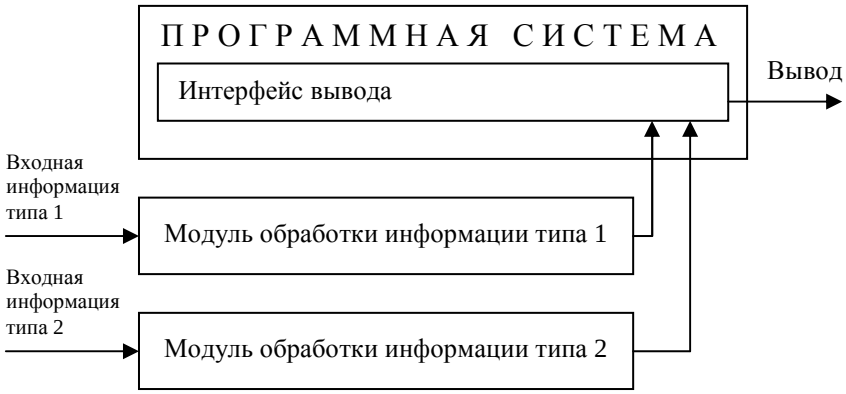

Рис. 7. Структура программной системы с внешними модулями обработки

Рис. 6. Алгоритм взаимодействия основной программы с подключаемыми модулями

Выводы и перспективы

Изложенный принцип создания программной системы воспроизведения параметрической информации с разделением функциональной нагрузки между основной программой и подключаемыми модулями позволяет избавиться от ряда недостатков существующих аналогичных систем. Задача воспроизведения параметрической информации посредством компьютера сводится к задаче создания специализированного подключаемого модуля, разработчики которого имеют возможность, используя ресурсы компьютера максимально адаптироваться к особенностям источника информации с минимальными ограничениями со стороны, воспроизводимой программы. За счет этого исключается проблема, связанная с расширением области применения системы и наращиванием количества подключаемых модулей (рис. 7). Программная система, реализуемая в рамках предлагаемой архитектуры, позволяет в простой и удобной форме воспроизводить различную параметрическую информацию и является эффективной заменой используемых в настоящее время аналогичных по функциональным характеристикам программных пакетов. Благодаря использованию гибкой настраиваемой основной программы во всех случаях данная система будет более простой и дешевой в разработке и сопровождении, а за счет использования внешних подключаемых модулей и предоставления средств для их создания сторонним разработчикам она может получить широкую область применения.

Дальнейшее развитие предполагает усовершенствование пользовательского интерфейса и наращивание функциональных возможностей основной программы в целом независимо от создания и модификации подключаемых модулей.

1. *Глобальный* аэронавигационный план применительно к системам CNS/ATM // Монреаль DOC. 9750 – AN/963 – 2002 – 325 с.
2. *Дмитриев Ю.В., Мишарин И.В., Пролыгин А.В., Яцков Н.А.* Объективный контроль полетов воздушных судов по информации бортовых регистраторов. – НАУ, 2007. – 112 с.
3. *Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts.* Refactoring: improving the design of existing code. Addison Wesley, 2000. – 432 p.
4. *Мишарин И.В., Непорожний Г.И.* Обработка и анализ полетной информации программно-аппаратным комплексом. – НАУ, 2007. - 160 с.
5. *Ластовченко М. М., Федченко С. В.* Концепция создания специализированного программного обеспечения для системы визуализации навигационной обстановки // Проблеми інформатизації та управління. – 2006. – № 1(16). – С. 93 – 97.