Подход к моделированию систем и сайтов из готовых ресурсов

Е.М.Лаврищева[1,2], **А.Г.Рыжов**[1]

[1]*Институт системного программирования им. Иванникова РАН*
[2]*Московский физико–технический институт*

Аннотация. Рассматривается компонентный подход к созданию систем и сайтов из готовых ресурсов (компонентов, объектов, сервисов и reuses). В основе подхода лежит графовая и компонентная модель (КМ), включающая функциональные, системные, сервисные и интерфейсные программные ресурсы и алгебру их обработки. Функции объектов-компонентов описываются в языках программирования (ЯП), а их интерфейсы в языках API, IDL, WSDL и др. Они реализуются в клиент–серверной архитектуре и взаимодействуют между собой через интерфейс. Дан анализ моделей систем, веб-сайтов и приложений, выполняемых в двухуровневой клиент-серверной архитектуре и представлен современный путь развития этой архитектуры в связи с увеличением числа пользователей, работающих с современными огромными объемами данных Big Data и Cloud Computing. Готовые ресурсы проверяются на правильность их спецификации в ЯП, тестируются и собираются сведения об ошибках для их исправления и оценки надежности. Отработанные ресурсы конфигурируются и система проверяются на функциональность. Описан сайт http://7dragons.ru, включающий готовые ресурсы и артефакты технологии разработки программных систем и новые сервисно–компонентные ресурсы (SOA, SCA, SOAP и др.), используемые для моделирования веб–систем и сайтов.

Ключевые слова: компонент, сервис, ресурс, система, веб–система, сайт, брокер сервисов, надежность, качество, Интернет.

Approach to the modeling of systems and sites from ready resources

Lavrischeva E.M. [1,2], **Ryzhov A.G.** [1]

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences*
[2] *Moscow Institute of Physics and Technology (State University)*

Annotation. The component approach to creation of systems and sites from ready resources (components, objects, services and reuses) is considered. The approach is based on the graph and component model (CM), which includes functional, system, service and interface software resources and algebra of their processing. Functions of objects-components are described in the programming languages (PL) and their interfaces in the languages of API, IDL, WSDL, etc. They are implemented in a client–server architecture and interact with each other through the interface. The analysis of models of systems, websites and applications running in a two-tier client-

server architecture are given and the modern way of development of this architecture is presented in connection with the increase in the number of users working with modern huge amounts of Big Data and Cloud Computing data. Finished resources are checked for the correctness of their specifications in the PL, and then errors information are tested and collected to correct the errors and assess their reliability. Spent resources are configured and the system is checked for functionality. Site described http://7dragons.ru, including ready-made resources and artifacts of software development technology, and new service-component resources (SOA, SCA, SOAP, etc.) used for modeling web-systems and sites.

Keywords: component, service, resource, system, web–system, site, service broker, reliability, quality, Internet.

Введение

Моделирование систем (Веб–приложений и сайтов) проводится с помощью моделей, элементами которых являются функциональные и системные элементы и их отношения между ними, образуя модель архитектуры системы. В 70–80 годы прошлого столетия сформировался подход к сборке разнородных программ и модулей в сложную систему с использованием интерфейсов при реализации комплексов программ специального и общего назначения в рамках ВПК (под руководством Липаева В.В.). Метод сборки был представлен в ОС ЕС (IBM–360) и в других общесистемных средах – OS Sun Microsystems, .Net, VS.MS, IBMSphere и др. Этот подход получил название – сборочного программирования [1–3]. По словам А.П.Ершова [4,5] «сборочное программирование обеспечивает построение из уже существующих (проверенных на правильность) готовых отдельных фрагментов, элементов (типа reuses) в сложную структуру». После 1994 года разработаны стандартные языки интерфейсов (MIL, IDL, API, WSDL и др.) и операции связи (make) готовых ресурсов в системах BSD, GNU, Microsoft, JEE, SPAROL, SOAP и др. [6–11]. Оператор make собирает исполняемые элементы из библиотек в Make File и задает им порядок связей друг с другом для разных сред OS Linux, .NET, IBMSphere и др. Метод сборки активно используется на фабриках программ (например, Д. Гриндфилд, К.Ленц – потоковая сборка программ, 2008; К. Чернетски – мультисборка на множестве языков, 2005; И. Бей – взаимодействие разноязыковых программ на современных языках, 2010 и др.). Нами предложена теория создания фабрик программ из готовых компонентов повторного использования – КПИ или Reuses [12–23]. Сегодня в связи с повышенным интересом многих стран к производству программных продуктов разного назначения, проблематика фабрик сборки получает большой интерес в информационном обществе. На статью [20] и сайт пришли десятки запросов с Китая, Таиланда, Южной Кореи, Канады и др.

После появления ООП (1987) был создан язык моделирования UML (Unified Modeling Language, 1994), а затем появился целый ряд других моделей (MDA, MDD, CIM, PIM, SOA, SCA и др.). На основе моделей создаются системы из готовых программных элементов, компонентов и проводится их

сборка в некоторый вариант программной системы (ПС). Компоненты описываются в современных ЯП – Java, Python, Ruby, C++ и др. Процесс сборки, интеграции программных элементов были стандартизированы (IEEE ISO 828–96–2012 Configuration) и позволяют получить конфигурацию модели архитектуры прикладной, программной и информационной системы для управления их выполнением в разных средах. В рассматриваемом подходе к моделированию систем, веб-приложений, веб-сайтов из готовых функциональных и сервисных ресурсов [1-12] используется стандарт конфигурирования вариантов систем из готовых ресурсов.

1. Сущность компонентного моделирования систем

Моделирование систем проводится с помощью графовой компонентной модели КМ, как составной модели метода ОКМ [12]. Компонент (Component, Comp, С) определяется как некоторая абстракция, включающая в себя интерфейсный раздел и артефакт функции. Он может иметь несколько реализаций в зависимости от операционной среды, модели данных, СУБД и др. [12–16]. Интерфейс определяет данные для связи одного компонента с другими. Компонент может иметь несколько интерфейсов. Компоненты наследуются в виде классов, используются в модели компонента и каркаса, описывается в любом ЯП. Компонентами могут быть:

1) программные, сервисные, системные и служебные;
2) серверные, клиентские и веб–серверные и веб–клиентские;
3) контейнерные, патерные и экземпляры классов;
4) системные (Big Data, Cloud Computing, Amazon и др.).

Каждый из этих типов компонентов имеет спецификацию, требования, правила их взаимодействия с другими компонентами. Для компонентов КМ разработана алгебра операций – добавления, удаления, замены и др. и компонентная алгебра определения свойств и характеристик моделей КМ (внешняя, внутренняя и эволюционная) [12, 13]. Приведем их.

Алгебра операций над компонентами

Операция добавления компонента С к компонентой среде обозначается $\oplus$. и выполняется согласно правилам $C \oplus CE_1 = CE_2$, $NameSpace = \{C.CName\} \cup CE_1.NameSpace$, $CE_2.InRep = \{C.(CIn_i, CName)\} \cup CE_1.InRep$, $CE_2.ImRep = \{C.(C_j, CName)\} \cup CE_1.ImRep\}$.

Аксиома 1. Операция добавления компонента к компонентной среде коммутативна и ассоциативна: $C \oplus CE = CE \oplus C$; $C_1 \oplus (CE \oplus C_2) = (C_1 \oplus CE) \oplus C_2$.

Операция удаления компонента из среды обозначается $\lozenge$ и имеет вид:

$CE_1 \lozenge C = CE_2$.

Теорема 2. Для любого компонента C в среде CE выполняется равенство $(C_2 \oplus CE) \lozenge C = CE$.

Операция замены компонента задается знаком "–" и имеет вид:

$CE.NameSpace(C_1) - C_2 = (CE \lozenge C_1) \oplus C_2$.

Операция объединения ($\cup$) компонентных сред имеет вид: $CE_1 \cup CE_2 = CE_3$,

Теорема 3. Операция объединения компонентных сред ассоциативна:

$(CE_1 \cup CE_2) \cup CE_3 = CE_1 \cup (CE_2 \cup CE_3)$.

Аксиома 3. Операция объединения компонентных сред коммутативна: $CE_1 \cup CE_2 = CE_2 \cup CE_1$.

Утверждение 2. Для любых компонентных сред выполняется равенство:$CE \cup FW = FW \cup CE$.

Утверждение 3. Для двух компонентных сред CE_1 , CE_2, и компонента *Comp* всегда выполняется: $Comp \oplus (CE_1 \cup CE_2) = (Comp \oplus CE_1) \cup CE_2 = (Comp \oplus CE_2) \cup CE_1$.

Операция добавления реализации, замены старой реализации новой и операции добавления и замены интерфейса имеют аналогичный вид и доказательство.

Операция добавления интерфейса и реализации компонентов

Операция *addImp Comp* добавляет новые входные интерфейсы и реализации Comp в среду или в модель: *NewComp = AddImp(OldComp, NewCImps,NewCIntOs*) и $NewInt = OldInt \cup NewIntOs$, где $NewCImp = OldCImp \cup \{NewImps\}(\exists\ OldIntt \in OldCIntI)\ Provide(OldInt) \subseteq NewImps$,

где *NewCImps* – новая реализация, которая добавляется;

OldCInt – множество существующих интерфейсов *Int*.

Условие целостности компонента выполняется автоматически, потому что множество входных интерфейсов остается прежним. Из целостности старого компонента вытекает целостность нового компонента.

Компонентная алгебра

Компонентная алгебра – это $\sum = \{\varphi_1, \varphi_2, \varphi_3\}$,

где $\varphi_1 = \{CSet, CESet, \Omega_1\}$ – внешняя алгебра;

$\varphi_2 = \{CSet, CESet, \Omega_2\}$ – внутренняя алгебра;

$\varphi_3 = \{Set, CESet, \Omega_3\}$ – алгебра эволюции.

Внешняя алгебра включает операции:

– инсталляция $CSet_2 = Cset \oplus CESet_1$;

– объединение компонентных сред $CESet_3 = CESet_1 \cup CESet_2$;

– удаление компонента из компонентной среды $CESet_2 = CESet_1 \setminus CSet$.

Внутренняя алгебра $\varphi_2 = \{CSet, CESet, \Omega_2\}$ включает операции:

– *addImp* – добавление реализации;

– *addInt* – добавление интерфейса;

– *replImp* – замена реализации компонента;

– *replInt* – замена интерфейса компонента.

Алгебра эволюции $\varphi_3 = \{Set, CESet, \Omega_3\}$ включает операции: рефакторинга; реинженеринга с заменой, переименованием компонентов и

интерфейсов и добавления новых компонентов в ПС; реверсной инженерии – восстановление исходной структуры компонента по выходному коду ПС.

Приведенные алгебры позволяют гибко управлять компонентами в моделируемой системе.

2. Моделирование систем и сайтов

Построение архитектуры (модели) системы и сайтов на компонентной основе состоит в представлении графовой модели, в вершинах которой размещаются отдельные функциональные элементы и сервисы, а дуги задают связи между ними [11]. Графовая модель объектов (ОМ) создается на уровнях проектирования с применением логико–математического аппарата:

– обобщающий уровень задает объект в виде денотата в соответствии с теории Фреге типа <имя объекта> <концепт>;

– структурный уровень задает теоретико–множественное упорядочение объектов и представления их в виде графа;

– характеристический уровень задает логико–алгебраическое определение характеристик объектов и их свойств;

– поведенческий уровень задает поведение и связи между объектами графа ОМ.

Согласно стандарту ISO/IEC 12207 Cycle Life процессы ЖЦ веб–систем начинаются с формирования и определения элементов системы:

– модель веб–системы и модель характеристик (Model Feature) для управления вариантами систем;

– веб–сервер и веб–клиент клиент–серверной архитектуры для выполнения запросов от клиента к серверу из веб-системы;

– операции связи и взаимодействия между программными элементами с помощью интерфейса;

– компоненты и сервисы с уникальными именами и интерфейсами;

– схемы запросов к функциям системы (например, протокол contract WorkFlow) с набором входных и выходных параметров.

Модель системы из компонентов

Модель ПС задается на множестве программных компонентов, заданных в графе. Вершины графа G (рис.1) соответствуют прикладным задачам и функциям $F=\{f_{Oi}\}$ предметной области (ПрО), а дуги задают связь компонентов через интерфейсы $I_{Oi} = \{Io_n\}$. Программный компонент с общей точки зрения, это самостоятельный продукт (код) метода или функции, который реализует часть или всю ПрО и взаимодействует с другими элементами системы через интерфейсы [12-15].

Модель системы имеет вид: $M_{ПС} = <M_f, M_s, M_i, M_d>$, где

$M_f = (f_{O1}, f_{O2},\ldots, f_{Or})$ – множество функций, каждая из которых реализует некоторый метод или задачу ПрО;

$M_s = (S_{in}, S_{out}, S_{inout})$ – множество входных данных S_{in}, выходных данных S_{out} и промежуточных данных S_{inoutt} на сервере Application и множество

системных сервисов (Common Facility Services) операционной среды, которые передаются брокером или монитором;

$M_i = (Io_1, Io_2, \ldots, Io_n)$ – множество интерфейсно–компонентных элементов, задающих для функциональных элементов f_{On} входные *in*, выходные параметры *out* и *inout* из множества M_s;

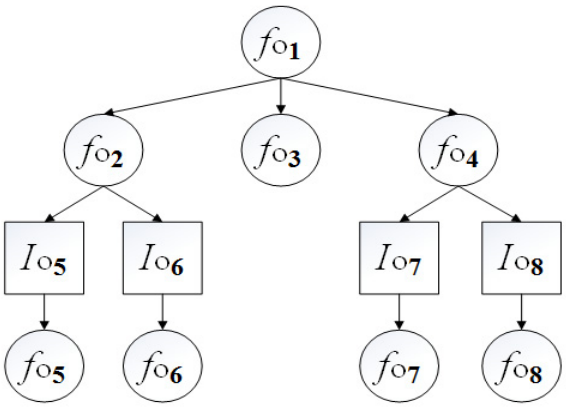


Рис. 1. Граф G на множестве функциональных, сервисных и интерфейсных объектов

$M_d = (M_{d1}, M_{d2}, \ldots, M_{dn})$ множество данных и метаданных ПрО, с которыми работает система, к ним относятся простые и сложные типы данные ЯП через интерфейсные посредники, задаваемые в IDL или WSDL.

Множества M_{nc}, M_s и M_i определяют готовые ресурсы, интерфейсы и общие данные. Они могут иметь пересечение по данным, которые задаются в *in*, *out*, *inout* и входят в состав внешних типов данных для обмена элементов модели через сервер Application, вычисляются на сервере соответствующими компонентами и полученные результаты передаются клиенту.

На основе графа G (рис.1) можно определить несколько моделей подсистем:

$M_{ПС1} = <F_1 (f_{O2}((Io_5, f_{O5}), (Io_6, f_{O5})); f_{O3}; f_{O4}((Io_7, f_{O7}), (Io_8, f_{O8}))>$,

$M_{ПС2} = <F_2 (f_{O2}((Io_5, f_{O5}), (Io_6, f_{O5})>$;

$M_{ПС3} = < F_3 (f_{O3})>$;

$M_{ПС4} = < F_4 ((Io_7, f_{O7}), (Io_8, f_{O8}))>$.

По этим моделям можно сконфигурировать отдельные версии выходной системы.

Теорема. Взаимодействие двух функциональных объектов является корректным, если первый объект полностью обеспечивает функции и передачу данных, необходимых другому объекту: $In(fo_k) \subseteq Out(fo_l)$.

Сайт системы – это набор готовых функциональных, системных и сервисных элементов, определяющих выполнение функций сайта. Вход на сайт осуществляет специальная программа Администратор, предоставляя пользователю, заданные на сайте проблемные и системные функции, а также принимает данные для выполнения программ сайта [16].

Модель сайта имеет вид: $M_{st} = <F_f, S_s, S_i, I_d, M_{sys}, M_d>$, где

$M_f = (f_{O1}, f_{O2,}\ldots, f_{Or,})$ – множество функций системы, каждая из которых реализует некоторый метод или задачу ПрО;

$M_{ser} = (S_{si}, S_{sout}, S_{snout})$ – множество сервисов для обработки входных, выходных и промежуточных данных на множестве системных сервисов (Common Facility Services);

M_{sys} – множество системных функций клиента и сервера веб–сайта;

M_d – множество данных и метаданных ПрО сайта, которые специфицированы как готовые ресурсы (сложные типы данных, объекты клиента, сервера и др.).

Системные элементы управляют оборудованием, ОС и веб–сервером. Веб–сервер производит обработку запросов или/и их генерацию, а также управляет выполнением элементов модели M_{st}.

Первые клиент–серверные архитектуры

Клиент–серверная архитектура была реализована в системе CORBA [13, 16-18] и обеспечивает взаимодействие ресурсов через:

– брокера объектных запросов (*Object Request Broker* — ORB) клиент–объектов с сервер–объектами на ЯП (Smalltalk, Cobol, Ada–95, Lisp, PL/1, C++, Python, Java, IDLScript и др.);

– общие объектные сервисы (*Common Object Services* — COS) для управления изменениями, реализациями, транзакциями, подпроцессами и т.п.;

– общие средства обслуживания (*Common Facilities* — CF) для объединения в различные конфигурации сервисных объектов;

– объектные приложения (*Application Objects* — AO), над которыми могут производиться операции – открыть, инсталлировать, переместить, поместить, выполнить.

Клиент и сервер обмениваются между собой с помощью запросов, каждый из которых обрабатывается брокером ORB на основе описания интерфейсов объектов для клиента, сервера.

Интерфейс клиента (*Client Interface*) обеспечивает взаимодействие с объектом–сервером и состоит из трех базовых интерфейсов:

– stub–интерфейса, содержащего описание внешних параметров и операций объекта в IDL;

– интерфейс динамического вызова (Dynamic Invocation Interface — DII) объекта, определяемого во время выполнения программы клиента при поиске интерфейса в репозитории интерфейсов или реализаций;

– интерфейса сервисов ORB (ORB Services Interface), содержащего сервисные функции для клиента и сервера.

Интерфейс передается через *stub* клиента серверу и возвращается результат через skeleton сервера. Интефейс DII работает во время выполнения. В каждом вызове указывается тип объекта, тип запроса и параметры. По этой информации извлекается программа из репозитория интерфейсов и реализаций [16-18, 22]. Описание интерфейса в IDL начинается с ключевого слова

interface, за которым следует: имя интерфейса, описание типов параметров и операций вызова объектов:

interface A { ... }
interface B { ... }
interface C: B, A { ... }

Описание типов данных начинается ключевым словом typedef, за которым следует базовый или конструируемый тип и его идентификатор. Типы данных описываются как фундаментальные типы данных ЯП: integer, boolean, string, float, char и др.

Пример реализованного Веб–сайта

В рамках клиент–серверной архитектуры построен сайт http://7dragons.ru – «Программная инженерия. Технологии и обучение SE» (ИТК) [16].

Данный сайт содержит разделы (рис.2 а), которые реализуют: **технологии** разработки ПС из готовых КПИ (сборка, конфигурация, онтология, качество, отображение данных и др.); **взаимодействие** между системными компонентами; инструменты поддержки сайта; **обучение** «Программной инженерии», ЯП JAVA, C#, C++, Basic, вычислительной геометрии и фабрикам программ. Разделы включают общее название технологии, семантику, реализацию и выполнение функций. Каждый раздел содержит подразделы, которые конкретизируют семантику подразделов (их 11). На рис. 2 б дано семантическое описание каждой функции из раздела 2 а.

В реализации данного сайта, отдельных линий, реализации технологий и компонентов разработки систем из готовых КПИ (ГОР) принимали участие магистранты МФТИ (А. Островский, И. Скотников), КНУ (И. Радецький и А. Аронов, А. Дзюбенко), аспиранты А. Колесник, А. Стеняшин, А. Рыжов и др. [19–24].

Для отображения структуры и содержания технологий была выбрана архитектура Model–View–Controller (MVC). Все страницы, отображающие статьи по тематикам сайта, строятся по единому шаблону и реализованы по одной схеме – **описание, пример, выполнение, выход.**

На этом сайте используются системные ресурсы:

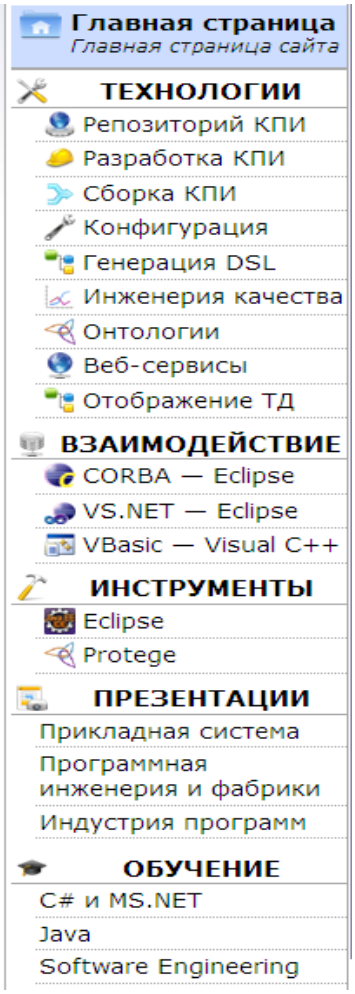
1) контейнер для управления экземплярами соединений с ИР;

2) экземпляры соединений с ИР, которые обеспечивают реализацию доступа к ресурсу;

3) компонент управления взаимодействием;

4) адаптер доступа к ресурсу;

5) дескриптор ИР, который задает информационную базу для проведения соединения и организации взаимодействия компонентов.



а)

Веб–сайт содержит разделы на рис. 2 *а* и перечень технологий на рис. 2 *б*.

ТЕХНОЛОГИИ ПОСТРОЕНИЯ ГОР и ПС:
Технология обслуживания репозитория КПИ,
Технология разработки КПИ,
Технология сборки КПИ,
Технология конфигурирования КПИ,
Технология генерации описания КПИ в языке DSL,
Технология оценка затрат и качества,
Технология онтологии вычислительной геометрии, ЖЦ ISO/IEC12207,
Технология веб–сервисов,
Технология генерация типов данных ISO/IEC 11404.

ВЗАИМОДЕЙСТВИЕ программ, систем и сред:
Модель CORBA – Eclipse–JAVA,
Модель VS.Net C# – Eclipse,
Модель Basic – C++.

ИНСТРУМЕНТЫ ИТК:
Система Eclipse,
Система Protégé

ПРЕЗЕНТАЦИИ ПС В ИТК:
Система ведения зарубежных командировок для НАНУ,
Слайды про ИТК, фабрики программ,
Методологии построения ТЛ.

ОБУЧЕНИЕ:
технологии программирования в C# VS.Net и JAVA, электронный учебник "Программная инженерия" веб–сайта.

б)

Рис. 2. Главная страница веб–сайта

Функции компонентов распределяет менеджер ресурсов следующим образом. Дескрипторы ресурсов задаются XML–файлами и располагаются на Веб–сервере. Менеджер – это контейнер экземпляров, каждый из которых реализует связь с определенным ресурсом и формирует дескриптор ресурса следующего вида:

```
<?xml version="1.0" encoding="WINDOWS–1251"?>
<ir_manager_description>
<example_name> Web–Сайт ISP
  <example_id>ir_manager_description_for_iss_site
  </example_id>
  <description> Web–Сайт ISP RAS
</description>
  <manager_type> web–site</manager_type>
  <location>
    <url>http://7dragons.ru/</url>
    <protocol>http</protocol>
  </location>
</ir_manager_description>
```

Генерация клиента и сервера на сайте

Генерируется два класса компонентов: клиент и сервер. Клиент задает интерфейс между программными компонентами, а пользователь использует их в таком виде:

```
…
int param;
…
ClientInterface ci = new ClientInterface();
int result = ci.func(param);
…
```

Здесь ClientInterface – это сгенерированный класс, который содержит механизмы, необходимые для передачи данных на сервер, который предоставляет сервис int func (int param).

Если сгенерированный класс – сервер, то кроме механизмов связи с клиентами в него помещают пустые функции, которые отвечают задекларированным методам в интерфейсе. Для использования сервера пользователю необходимо реализовать конкретную логику данных методов, которая имеет вид:

```
class ServerInterface {
    …
    public int func (int param){
            …//реализация метода
    }
}
void main(){
    …
    ServerInterface si = new ServerInterface();
    si.work();
    …
}
```

Эти примеры демонстрируют обеспечение функциональности и связи разноязычных программ через клиента и сервера.

Обеспечение качества и надежности систем

Качество – это совокупность свойств (показателей качества) ПО, которые обеспечивают его способность удовлетворять потребности заказчика, в соответствии с его назначением. Оно задано в стандарте ГОСТ 2844–98 с помощью модели качества и его показателей. Стандарт ISO/IEC 12207 определяет основные процессы ЖЦ системы и дополнительные процессы, которые регламентируют планирование, управление качеством и оценку затрат на проект [13]. На этапах ЖЦ проводятся операции на анализ качества ПО:

- достижение качества ПО в соответствии с требованиями и критериями;
- верификацию и аттестацию (валидацию) промежуточных результатов ПО на этапах ЖЦ и измерение степени достижения отдельных его показателей;

• тестирование готовой ПС, сбор данных об ошибках, дефектах и отказах в системе для оценивания надежности и других показателей качества.

Модель качества ПО согласно стандарту ISO/IEC 9126 включает шесть показателей q_1–q_6 (q–quality) качества:

q_1 – функциональность (functionality),
q_2 – надежность (realibility),
q_3 – удобство (usability),
q_4 – эффективность (efficiency),
q_5 – сопровождаемость (maitainnability),
q_6 – переносимость (portability).

Каждый показатель качества q_i оценивается по специальной формуле и метрикам стандарта. Одним из показателей качества является надежность. Для оценки надежности веб–систем используются данные об ошибках, дефектах и отказах, собранные на этапах проектирования, верификации, тестирования и во время опытной эксплуатации. Надежность является функцией от обнаруженных ошибках и отказах в системе. Оставшиеся необнаруженные ошибки проявляют себя время от времени при определенных условиях (например, при некоторой совокупности данных) функционирования системы.

В зависимости от собранных типов ошибок выбирается модель надежности и проводится измерение показателя надежности [23].

Данные по всем показателям качества q_1–q_6 оцениваются по формуле:

$$q_1 = \sum_{j=1}^{6} a_{1j} m_{1j} w_{1j}$$

где a_i – атрибуты каждого показателя качества (i=1–6); m_i – метрики каждого атрибута качества; w_i – вес каждого атрибута показателя качества системы. Полученные данные по характеристикам (показателям) модели качества входят в сертификат качества.

Расчет стоимости ПС (Модель COCOMO 2)

Номинальные трудозатраты на разработку ПС (средние) рассчитываются по формуле:

$$T_{\text{ном}} = A \cdot V^B$$

где A = 2,45 – константа, полученная при анализе 80 реальных проектов, V – предполагаемый размер ПС или ПО в тысячах строк исходного кода KSLOC, B – показатель степени при размере V, учитывающий эффективность процесса разработки:

$$B = 0.91 + 0.01 \sum_{j=1}^{5} \Phi_j$$

где Φ_j – значения соответствующих коэффициентов атрибутов оценок $T_{ном}$ в других единицах измерения.

Номинальная продолжительность разработки ПО оценивается так:

$$Д_{\text{ном}} = 3.67 \cdot T_{\text{ном}}^{0.38+0.2(B-1.01)}$$

Тестирование на сайте

ИТК сайта обеспечивает изготовление готовых ресурсов, ПИ, сборку и тестирование объектов и компонентов в VS.Net, Java, Eclipse и др.[16].

Тестирование конфигурационной модели сборки проводится с помощью набора тестов для отдельных элементов и системы в целом. Используется метод тестирования Дж. МакГрегора [27] «от требований» (requirements–based testing). Тесты проверяют функциональные и интерфейсные объекты. В качестве инструмента тестирования используется фреймворк Visual Studio 2007 со средствами проверки правильности тестирования разных видов объектов. *Test Manager* управляет средствами планирования процесса тестирования и выполнения тестовых сценариев. При обнаружении ошибок в процессе тестирования вносятся исправления в модели системы M_{nc}. Затем проводится повторная конфигурация и получение готового продукта после тестирования.

Базовые средства создания сайта ИТК

На сайте ИТК использовались базовые средства реализации сайтов[16]:

- **DTP (Data Tools Platform)** — системы управление данными (data–centric systems) с большим числом конекторов;
- **GEF (Graphical Editing Framework)** — фреймворк для построения встроенных графических редакторов;
- **Jazz** — клиент-серверная платформа надстраиваемая поверх Eclipse (входит в линейку продуктов IBM Rational);
- **EMF** — среда моделирования Eclipse для создания моделей и генерации кода в XMI;
- **UML2** — метамодели UML 2.0 для моделирования систем;
- **Aspect** — аспектно–ориентированное расширение Java, C/C++IDE;
- **PDT (PHP Development Tools)** — среда разработки на PHP;
- **TPTP (Test & Performance Tools Platform)** —дебаггеры, профайлеры тестирования и HTML, JavaScript, CSS, JSP, SQL, XML, DTD, XSD, WSDL и др.

Данный сайт http://7dragons.ru используется в МФТИ для получение знаний о технологиях, решении отдельных задач и языку Java (Хибабулина), C#, VS.Net и предмета «Программная инженерия».

4. Формализация модели клиента и сервера веб–сайтов

Основу веб–сайта в Интернет составляет Интернет–браузер [3, 12, 16, 18], архитектура клиента, веб–сервера и База сайтов (репозиторий и БД).

Архитектура клиента – это Интернет–браузер, включающий характеристики:

T^1 = {Chrome, Firefox, MS Internet Explorer, Safari, Opera, …}.

Архитектура сервера – это веб–сервер, в среде которого реализуются компоненты обработки запросов клиента и генерации ответов сервером. То есть веб–сервер – это компонент веб–сайта, который инкапсулирует все операции клиента при обработке запросов и включает характеристики систем:

T^2= {Internet Information Server, Apache, JBoss, WebSphere, WebLogic, Cloudscape}.

Интерфейс между клиентом и сервером веб–системы определяется совокупностью запросов клиента в CGI и имеет вид:

WebAppInterface={Request^p}, где *Request^p* – p–й запрос.

Общая структура запроса клиента имеет вид: Request^p=(URL^p, ParamList^p), где интерфейс между клиентом и сервером определяется протоколом вида:

WebAppInterface={Request^p}, где *Request^p* – p–й запрос.

Общая структура запроса клиента имеет вид: Request^p=(URL^p, ParamList^p), где *URL^p* – сервер, который получает запрос с параметрами *ParamList^p={(ParamName^{pq}, ParamValue^{pq})}*; *ParamName^{pq}* – имя q–го параметра p–го запроса, а *ParamValue^{pq}* – его значение.

Некоторые параметры запроса могут отсутствовать и тогда *ParamList^p* = **Ø.**

Клиент веб–сайта. Архитектура клиента строится на основе типовых моделей, объектов данных, операций Интернет–браузера и состоит из нескольких логических уровней для передачи данных серверу с целью обработки и получения результата. Клиент имеет характеристики:

T^3={объекты данных, функциональные компоненты, отображенные данные}.

Такие элементы задаются в одном из скриптовых ЯП браузера и обеспечивают:

– работу с массивами, таблицами, списками;

– верификацию данных, которые вводит конечный пользователь;

– работу с нестандартными окнами браузера для отображения объектов данных.

Компоненты клиента описываются средствами ЯП:

T^4 = {JavaScript, VB Script, Java, HTML, XML}.

Сервер веб–сайта. Для описания серверных компонентов используется ЯП с характеристиками сервера MS Internet Information Server такого вида [14-17]:

T^5 = {ASP, JavaScript (серверный), VB Script (серверный), C, C++, ASP.NET, C#, C++ .NET, VS .NET, J# .NET, XML, SQL}.

Веб–сервер APACHE обеспечивает функционирование компонентов в разных ЯП и их характеристики имеют вид:

T^6_{APACHE} = {PERL, PHP, PYTHON, XML, SQL}, а для Java–среды (Tomcat, JBoss, WebSphere, WebLogic и др.) имеет вид:

T^6_{JAVA} = {JAVA, JSP, JSTL, JSF, XML, SQL}.

К компонентам сервера веб–сайта относятся:

T^7 = {прикладные компоненты веб–сервера, компоненты управления обработкой запроса, операции обработки данных, описание данных}.

Компоненты сервера имеют типовый шаблон проектирования MVC:

T^7 ={контролер, серверные страницы, функциональные компоненты}.

Компоненты уровня управления обработкой запроса имеют такую характеристику

T^8 = {анализ, обработка и генерация ответа}.

При обработке запроса используются характеристики:

T^9={отбор и обработка данных, формирование результатов}.

Эквивалентом таких компонентов являются компонентная модель EJB, которая используются при генерации ответа клиенту. Операции доступа к данным и их хранения в информационном ресурсе (ИР) имеют следующие характеристики:

T^{10}={модели данных, описание доступа, операции обработки данных}.

Эти компоненты связаны с данными, находящимися в ИР (базы данных, документов, файлов и др.), обеспечивают доступ к данным и взаимодействие с ресурсами типа entity – в модели EJB.

Совокупность характеристик $T = \{T^1, T^2, T^3, T^4, T^5, T^6_{IIS,,APACHE}, T^6_{JAVA}, T^7, T^8, T^9, T^{10}\}$ определяет систему классификации компонентов для веб–сайтов.

Модели Веб–систем Интернет

В настоящее время в Интернет накоплено огромное количество готовых ресурсов типа сервисов. Модели SOA (Service Oriented Architecture) и SCA (Service Component Architecture) дают механизмы моделирования веб–систем и сайтов из сервисных и готовых reusable компонентов сервера Интернет, находящихся в библиотеках или репозиториях. Элементы SOA задают описание некоторой функции (Function) с заданным качеством сервиса (Quality service) в IT–стандартах комитета W3C. SCA – это архитектура из сервисов и компонентов, которые могут быть разработаны различными организациями, как готовые компоненты типа reusable. К ним относится EJB сервер приложений J2EE, сетевые сервисы, объекты планирования, доступа к БД и к системе. Эта архитектура включает удаленные КПИ и reuses, которые обмениваются между собой гетерогенными данными из множества общих сетевых сервисов Интернет. Элементы архитектуры могут собираться в систему путем интеграции или конфигурационной сборки.

Исходя из моделей систем и SCA среды Интернет, проектирование веб–систем начинаются с моделирования архитектуры, включая:

- модель веб–системы для управления вариантами систем;
- клиент–серверную архитектуру с веб–сервером и веб–клиентом для выполнения запросов от клиента к серверу из модели веб–системы;
- интерфейсы взаимодействия между компонентами;
- компоненты, сервисы и интерфейсы помечаются уникальными именами для передачи данных;
- схема запроса к имени функции или компонента с помощью операторов вызова или протоколов связи стандарта ISO, в которых описываются входные и выходные параметры.

Сервисы для описания веб–систем

Наряду с компонентным подходом широкое распространение получил сервисный подход. С его помощью реализуются веб–системы из готовых системных, функциональных и прикладных компонентов [3, 6, 7, 12–19].

Главная идея сервисного подхода Интернет и Semantic–Web состоит в том, чтобы накапливать независимые сервисные компоненты и собирать их для разных задач. Функциональные и прикладные сервисы в Интернет – это обычные программные ресурсы, которые реализуют отдельные задачи (в математике, промышленности, экономике, авиации и др.) и накапливаются в веб–библиотеках Интернет. Они обладают способностью к взаимодействию с помощью локальных и/или глобальных сетей Интернет и задаются в языках ЯП, а их интерфейсы в IDL, API, WSDL и др.

К сервисам относятся [6, 7, 14, 15, 16]:

– общие сервисы системных сред (J2EE, .Net, Appach, JAVA и др.), устанавливают связи с другими сервисами и компонентами через механизмы вызова RPC, RMI с помощью служб именования, каталогизации и др.;

– сетевые сервисы стандартной модели OSI, API, модели SOA, SCA задают обработку сервисов в сети Интернет.

Готовые программные и сервисные ресурсы (services, artifacts, reuses, assets и др.) используются как многоразовые КПИ для решения разного рода вычислительных задач, бизнес задач и др. Некоторые из сервисов стали обязательной частью общесистемных средств (VS.Net, IBM, Intel, Linux и др.), а также используются в специальных областях знаний (медицина, биология, генетика, геофизразведка и др.) [29].

Модель сервисов SOA

К основным моделям сервисов Интернет относятся SOA, SCA, SOAP (Service–oriented Application Program) [3–15].

Модель SOA – это набор принципов и средств создания системного ПО и прикладных систем с помощью совместимых и унифицированных сервисов Интернет. SOA задает реализацию сервисов на серверной стороне. Сервис содержит открытый интерфейс с описанием типов входных/выходных параметров в языке WSDL и портов обмена метаданными (Metadata Exchange Endpoints). WSDL–компилятор текст описания в этом языке готовит для сервера и клиента в виде прокси–классов. Они учитывают особенности конкретных программных платформ и ЯП. SOA обеспечивает согласованность, языковую независимость и интероперабельность серверной и клиентской частей системы. От разработчика требуется написать сервис средствами WCF (Window Communication Foundation) и использовать его в ЯП (Java, Python, Ruby и др.). Клиенты имеют на своей стороне заместители сервера прокси–классы.

Фундаментом сетевых служб SOA являются:

1. Набор языков – XML, SOAP, UDDI, WSDL, BPREL, BPMN и др. для реализации базовых свойств сетевых сервисов, обеспечения их взаимодействия между собой в соответствующих средах (SOA, SCA и др.);
2. Поставщик услуги, который осуществляет ее реализацию в виде сетевой службы, прием и выполнение запросов пользователей, а также публикацию сведений о сервисе в реестре;

3. Реестр (каталог) служб содержит библиотеку сервисов, а также средства их поиска и вызова с помощью запросов, которые поступают от поставщиков сервисов на получение сервисов;

4. Пользователь или потребитель сервиса (приложение, компонент и др.), который осуществляет поиск и вызов необходимого сервиса из реестра описания сервисов, также использует сервис, предоставляемый провайдером в соответствии с заданным интерфейсом.

Связь между поставщиком и потребителем осуществляется через HTTP и XML–сообщения сетевой среды, которая использует интерфейсы веб–сервисов. Посредником между этими сервисами и системой является *провайдер*, который обеспечивает взаимодействие между поставщиками и провайдерами с помощью средств описания и передачи сервисов WSDL, SOAP, XML.

Модель SCA

SCA – это набор программных компонентов сервисного типа и средств создания из них прикладных систем и сайтов. Сервисно–компонентная архитектура SCA [21] предназначена для работы с прикладными компонентами с разными спецификациями, разработанными различными компаниями и включает: EJB сервер приложений J2EE компании Sun Microsystems, который работает с сетевыми сервисами, компонентами доступа к БД и к ИС предприятия (Enterprise Information System, EIS) и др.

SCA обеспечивает доступ к сервисным компонентам и определяет зависимости между ними через аппарат ссылок. Компоненты SCA системы IBM WebSphere Integration Developer (WebSphereID) могут быть упакованы в модуль для выполнения сервисного модуля с WebSphere Process Server – эквивалентного EAR–файлу J2EE и некоторым другим. Подмодули J2EE и артефакты упаковываются с модулем SCA, что позволяет запустить сервис через модель SCA и передавать данные для обработки и интеграции.

Функция Dynamic Profiles WebSphere Portlet Factory обеспечивает динамическую конфигурацию пользовательского интерфейса.

Сервисы уровня на предприятии запускаются на узле сервисов предприятия, который содержит продукты, обеспечивают сервисы данных, безопасность и другие службы.

Сервер каталогов Tivoli (Tivoli Directory Server) обеспечивает протокол доступа к каталогам LDAP (Lightweight Directory Access Protocol) для управления идентификацией. Реестр WebSphere Service Registry and Repository позволяет провайдерам регистрироваться, а клиентам – выбирать сервисы.

Сервисно–компонетная модель SCM представляет собой обобщение объектно–компонентной модели продуктов (ОКМ, [16, 25, 26]). В ней каждый программный элемент содержит удаленные компоненты reuses – КПИ, которые обмениваются гетерогенными данными и обеспечивают выполнение системы. При этом используются механизмы сервисных объектов данных SDO и сервисы доступа DAS.

Модель SCM имеет вид: $SCM = \{N, ID, MD(N), MI(ID), SR\}$,

где *N* – имя системы,

ID – идентификатор модели SCA;

MD(*N*) – подмодель *абстрактных сервисов,* отображающая потребности системы в функциях/данных на множестве бинарных отношений, которые задают взаимосвязи между ними;

MI (ID) – *интерфейсная* подмодель на множестве интерфейсов, которая реализует абстрактные сервисы на множестве бинарных отношений и задает взаимосвязанные интерфейсы;

SR= (*R, RR)* – подмодель *сервисных ресурсов MI* (*ID*) на множестве *R* ресурсов и *RR* бинарных отношений, которые задают взаимосвязи между этими ресурсами. Элементы множества *R* – сервисы удаленных КПИ, сетевые сервисы и *композиты* (composites) SCA объектов специального типа соединяют свойства КПИ и сетевые сервисы.

Для описания сетевых сервисов в языке WSDL вводятся следующие виды описаний:

– строка (xsd:string),

– целые числа (xsd:int, xsd:long, xsd:short, xsd:integer, xsd:decimal),

– числа с плавающей запятой (xsd:float, xsd:double),

– логический тип (xsd:boolean),

– последовательность байтов (xsd:base64Binary, xsd:hexBinary),

– дата и время (xsd:time, xsd:date, xsd:g),

– объекты (xsd:anySimpleType).

Протокол Contract WorkFlow задает контракт-протокол для связи клиента и сервера в среде VS.Net.

WCF содержат три вида контрактов:

1) сервисов для описания функциональных операций, реализованных сервисом. Внутри контракта сервиса имеются контракты об операциях, как отдельные операции сервиса, которые реализуют функции;

2) данных, определяющих формат данных, которыми будут обмениваться сервисы. Это относится как к запросу на сервис, так и к октету сервиса.

3) сообщений, как тип контракта, который используется для того, чтобы получить контроль над заголовком SOAP.

Пример описания сообщения в языке XML в WCF:

<?xml version="1.0" ?>

<env:Envelope xmlns:env="

http://www.cbsystematics.com">

<!–Конверт протокола SOAP–> <env:Header>

<!–Заглавие протокола SOAP–> </env:Header> <env:Body>

<!–Тело протокола SOAP–> </env:Body> </env:Envelope>

При написании контрактов WCF атрибутами будут [ServiceContract], [OperationContract], [FaultContract], [MessageContract] и [DataContract].

На этапе выполнения клиента вызывается метод, определенный в интерфейсе сервиса, WCF сериализует типы CLR и вызов метода в формат XML

и посылает сообщение в сеть для привязки к схеме кодировки в WSDL. Со стороны XML задается XSD–описание структуры данных и сообщение осуществляется лишь после того, как будет создан экземпляр XML (XML Instance). Со стороны .NET имеется тип CLR, который определяет структуры данных и функциональные возможности после того, как создан объект такого типа.

Сервисы поддержки веб–систем в Интернет

Для сборки (композиции) сервисов используется инструмент – ***Jopera for Eclipse*** (http://www.jopera.ethz.ch/), который обеспечивает [18, 23– 26]:

– композицию сервисов (типа Agile) и визуальный мониторинг отладки композиций сервисов;

– управление изменением интерфейсов сервиса с помощью сообщений об изменениях в сервисе Jopera;

– масштабируемость и автономное исполнение процесса запуска систем с помощью сообщений Jopera.

Jopera предоставляет набор Eclipse–плагинов для связи различных программных элементов и допускает итеративную композицию сервисов (через маршрутизаторы SOAP и RESTful веб–сервис, Grid–сервисы, Java snipets и др.), а также путем моделирования и исполнения процессов в сети. Для поиска сервисов по их семантическим описаниям используются *Feta Client* и *Feta Engine*.

Feta Client – это GUI–плагин системы Интернет Taverna, используемый для описания сервиса, а *Feta Engine* для задания Веб–сервиса.

Подключаясь к *Feta Engine*, плагин *Taverna Feta* позволяет:

– конструировать ориентированные на заданную предметную область семантические запросы к нужным сервисам, которые затем отсылаются на *Feta Engine*;

– отображать информацию о результатах выполнения запроса на поиск сервисов;

– интегрировать результирующие сервисы в системе WorkFlow.

Критериями поиска сервисов являются:

– сервис, на входе которого находится элемент семантического или общего типа X;

– сервис, который производится на выходе системы и выдает элемент семантического типа Y;

– сервис, который решает задачу X или еще более конкретную;

– сервис, который использует метод X и более конкретный;

– сервис процессора WSDL и др.

– сетевые сервисы стандартной модели OSI, SOA, SCA, как инструменты представления и обработки ресурсов в сети Интернет для реализации деловых, финансовых, экономических и других услуг при решении разных прикладных задач.

Для разработки Веб–систем в Семантик Веб используются средства [25, 26]:

RDF стандарта W3C (2004) для описания сетевых, семантических ресурсов и метаданных (данные о данных). Служит каркасом для создания отдельных компонентов семантической паутины. RDFS (англ. *RDF Schema*) — это надстройка над RDF, которая позволяет создавать классы и свойства объектов.

OWL (Web Ontology Language) построен на форматах RDF и RDFS, предназначен для описания онтологий, логики и согласуется с современными сетевыми стандартами.

SPARQL (Protocol And RDF Query Language) — язык запросов для быстрого доступа к данным RDF для получения необходимой информации из сети:

RIF – формат обмена правилами (Rule Interchange Format) и др.;

WSDL – язык описания входных и выходных данных для описания запросов Интернет на сервисы и включает языки:

- WSCI (Web Services Choreography Interface [25]),
- WSCL (Web Services Conversation Language [26]),
- BPMN (Business process and model and notation [25]),
- BPEL (Business Process Execution Language for Web Services [26]) и другие.

В качестве адреса объектов в сети используются универсальные идентификаторы ресурсов URI (Uniform Resource Identifier) и интерфейс, задаваемый для управления связями с другими сервисами через XML–документы.

Конфигурация ресурсов веб–систем и сайтов

Под конфигурацией системы понимается структура некоторой версии системы, включающая функции, объединенные между собой операциями связи с параметрами, задающими режимы функционирования системы [16–19]. Выпуск версии вариантов системы осуществляется для поставки заказчику.

Версия или конфигурация системы согласно IEEE Standard 828–2012 (Configuration) включает:

– базис конфигурации – ВС (Configuration Baseline);

– элементы конфигурации (Configuration Item);

– программные компоненты, входящие в описание моделей ПС;

Управление конфигурацией (Configuration Management) заключается в наблюдении за модификацией параметров конфигурации и компонентов системы, а также в проведении систематического контроля, учета и аудита внесенных изменений, поддержки целостности и работоспособности системы.

Управление конфигурацией состоит в выполнении следующих основных задач:

1. Идентификация конфигурации (Configuration Identification).
2. Контроль конфигурации (Configuration Control).
3. Учет статуса конфигурации (Configuration Status Accounting).

4. Аудит конфигурации аудит (Configuration Audit).

5. Трассировка изменений конфигурации на этапах сопровождения и эксплуатации системы;

6. Согласование между собой объектно–компонентных и вариабельных моделей характеристик ресурсов и ПС;

7. Доказательство изоморфного отображения компонентных моделей систем и последовательная трансформация методов объектов и данных.

При конфигурационной сборке готовых ресурсов используется модель системы и модель характеристик MF. Готовые ресурсы и КПИ накапливаются в репозиториях или библиотеках системы. Они отбираются, адаптируются и интегрируются в единую систему. Основную роль в этих процессах выполняет конфигуратор ИТК. Он обеспечивает связь разнородных ресурсов и их интерфейсов с вариантами отдельных рабочих продуктов, которые находятся в репозитории.

Для управления артефактами системы создается *модель среды* конфигуратора, в которую входят:

– процесс сборки КПИ и артефактов системы;

– схемы формального описания артефактов;

– модели вариантов системы;

– конфигуратор артефактов и КПИ.

При управлении конфигурацией собираются данные для проведения таких стандартных операций, как *отчетность и аудит конфигурации* на установление запланированной функциональности системы. Конфигуратор собирает требуемые артефакты и КПИ в структуру веб–системы и создает конфигурационный файл для выполнения. в операционной среде.

5. Современные направления развития клиент-серверной архитектуры

В связи с возникающими атаками и различными угрозами в Интернет нами предусматриваются средства обеспечения безопасности, защиты данных и оценки качества как отдельных функциональных, системных и сервисных компонентов, так и самой веб–системы или веб–сайта [6-13, 28].

С учетом этих вызовов, возрастающим количеством пользователей сети и большим объемов данных (Big Data) предлагается усовершенствование рассмотренной клиент–серверной архитектуры в направлении формирования множества отдельных серверов для хранения и обработки больших данных. Это позволяет оптимизировать нагрузки на ресурсы, сетевое оборудование и более эффективно использовать возможности серверов и клиентов сети. Увеличение количества доступных ресурсов проводится путем вертикального и горизонтального масштабирования. Вертикальное масштабирование увеличивает производительность приложений за счет добавления аппаратных ресурсов сервера, а горизонтальное масштабирование способствует росту производительности приложений за счет увеличения количества серверов.

Такая клиент–серверная архитектура основывается на двух процессах: **Front–end** для клиентского приложения и **Back–end** для обслуживания серверной части приложения. Эта архитектура включает три компонента: клиент веб–браузера, сервер приложений с обеспечением надежности и сервер баз данных с защитой данных.

Front–end – клиентская сторона пользовательского интерфейса к программно–аппаратной части веб–приложения. В функции Front–end входит обработка соединений со всеми пользователями приложений, а также шифрование ответов при включенном SSL сертификате (Secure Sockets Layer – криптографический протокол с безопасной связью), авторизацию пользователей, ответов Back–end и т.п. Для выполнения этих задач веб–приложений понадобятся высокопроизводительные процессоры. Front–end сервера обеспечивает передачу пользователю данных большого размера (например, медиа–контент: фотографии, аудио– и видеоинформация). Front–end сервер определяет по пользовательскому URI (Uniform Resource Identifier, унифицированный идентификатор ресурса), на каком из серверов лежит требуемый файл и перенаправляет запрос на тот сервер, где располагается файл, и этот сервер отдает файл со своего жесткого диска.

На Back–end серверах запускается серверное приложение на конкретном ЯП (Python, Ruby, Java, PHP и т.п.) и на них сосредоточивается большая часть логики приложения. Вне этого уровня остаются только элементы, связанные с уровнем клиента, а также триггеры и хранимые процедуры, которые находятся на сервере БД. Back–end сервера проектируются таким образом, чтобы добавление к ним дополнительных серверов обеспечивало горизонтальное масштабирование производительности программной системы (комплекса) и не требовало внесения изменений в программный код приложения.

Данный подход ориентирован на работу с большими данными и допускает выполнение большого количества различных пользовательских задач.

Заключение

Данный подход к созданию систем и сайтов развивается в рамках проекта РФФИ №16–01–00352–18 «Теория и методы разработки изменяемых программных и операционных систем» [6-15]. В работе рассмотрены базовые понятия готовых ресурсов (компонентов, объектов, сервисов), моделей систем и метода сборки. Они специфицируются в языках (C++, JAVA, Python, Basic и др.). Описан компонентный метод, основу которого граю ОМ и компонентная модель. Дано формальное описание компонентного и сервисного моделирования веб-систем и сайтов в клиент-серверной архитектуре. Сформулированы формальные характеристики клиента и сервера в структуре систем, веб–системы и сайта. Приведена сущность моделей SOA, SCA и SCM для представления сервисов, серверов и клиентов управления сервисами. Представлено описание реализованного сайта http://7dragons.ru/ для создания систем из готовых ресурсов (компонентов, объектов, сервисов и reuses) и для

обучения языкам программирования Java, Basic и курсу «Программная инженерия». Рассмотрено современное развитие клиент–серверной архитектуры с множеством серверов, предназначенных для работы с большими данными и приложениями в условиях безопасности, надежности и качества программно–аппаратного комплекса.

Литература

1. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование //Киев: Наук. думка, 1991. - 236 с.
2. Липаев В.В., Позин Б.А., Штрик А.А. Технология сборочного программирования. М.: Радио и связь, 1992.
3. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов // Киев: Наук. Думка, 2009 – 371 с.
4. Ершов А.П. Опыт интегрального подхода к актуальной проблеме ПО. Кибернетика, 1984. - с. 11-21.
5. Ершов А.П. Научные основы доказательного программирования. Доклад АН СССР, 1985. - с. 1–14.
6. Лаврищева Е. М., Карпов Л. Е., Томилин А. Н. Подходы к представлению научных знаний в Интернет науке. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017. - с.310-326.
7. Лаврищева Е. М., Карпов Л. Е., Томилин А. Н. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей// Научный сервис в сети Интернет: труды XVIII Всероссийской научной конференции (19-24 сентября 2016 г., г. Новороссийск). — М.: ИПМ им. М.В. Келдыша, 2016. - с.126-138. - doi:10.20948/abrau-2016-8.
8. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation, 27 April 2007. https://www.w3.org/TR/SOAP12-part1
9. Web Services Description Language (WSDL) 1.1//W3C Note 15 March 2001. https://www.w3.org/TR/wsdl
10. Reference Model for Service Oriented Architecture 1.0. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html
11. Web Services Resource 1.2. WS-Resource - http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf, WS-Resource Properties - http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.
12. Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН, № 29, 2016 г. - М.: 48 с. ISBN 078-5-91474-025-9.
13. Лаврищева Е.М. Компонентная теория и коллекция технологий для разработки индустриальных приложений из готовых ресурсов, Труды 4-ой научно-практической конференции «Актуальные проблемы системной и программной инженерии», АПСПИ-2015, 20-21 мая 2015, с. 101-119.

14. Лаврищева Е.М., Петренко А.К. Моделирование систем и их семейств. Труды ИСП РАН, 2016, том 28. вып. 6. - с. 180-190. DOI: 10.15514/ISPRAS-2016-28(6)-4.
15. Кулямин В.В., Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды Института системного программирования РАН, том 28, вып. 3, 2016, стр. 189-208. DOI: 10.15514/ISPRAS-2016-28(3)-12.
16. Лаврищева Е.М. Программная инженерия. Парадигмы, Технологии, CASE-средства Программирования. 2 изд. – М.: Юрайт, 2016. – 280 с.
17. Лавришева Е.М., Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС. – М.: Финансы и статистика. 1982 - 136 с.
18. Лаврищева Е.М. Программная инженерия и технология разработки программных систем. Юрайт, 2016 - 431 с.
19. Островский А.И. Подход к обеспечению взаимодействия программных сред JAVA и MS.Net. - Проблемы программирования, - 2011. №2. с. 37-44.
20. Lavrischeva K.M. Theoty and Practice of Software Factories. - Cybernetic and Systems Analyses, 2011.Vol. 47.-N6. - p. 961-972.
21. Лаврищева Е.М., Колесник А.Л., Стеняшин А.Ю. Объектно-компонентное проектирование программных систем// Теоретические и прикладные вопросы. – Весник КНУ, серия физ.-мат. наук. – Киев, 2013. – №4. – С. 150–164.
22. Ekaterina Lavrischeva, Andrey Stenyashin, Andrii Kolesnyk. Object-Component Development of Application and Systems. Theory and Practice /Journal of Software Engineering and Applications, 2014, http://www.scirp.org/journal/jsea.
23. Ekaterina M. Lavrischeva. Assemblling Paradigms of Programming in Software Engineering.- 2016, 9, 2016. - pp.296-317, http://www.scrip.org/journal/jsea, http://dx.do.org/10.4236/jsea.96021.
24. Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle, "Science and Information Conference-2015", Jule 28-30, London, UK, www.conference.thesai.org.- p.965-972.
25. Semantic Web programming. S. Hebeler, M. Fisher, R. Blace, A. Peter-Lopes, Willey Publishing Inc., 2008. http://semwebprogramming.org/
26. Semantic Web, Representation of data on the World Wide Web, based on the RDF standards. http://www.w3.org/2001/sw/
27. MacGregor S.D., Sykes D.A. Practical Guide to testing Object-oriented Software. Addison-Wesley Professional, 2001.
28. Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов. Труды ИСП РАН, том 30, вып. 2, 2018, DOI: 10.15514/ISPRAS-2018-30(3).
29. E.M. Lavrischeva, I.B. Petrov.Ways of Development of Computer Technologies to Perspective Nano, Future Technologies Conference (FTC-2017) , 29-30 November 2017. Vancouver, Canada.-p.539-547.

References

1. Lavrischeva E.M., Grishchenko V.N. Sborochnoe programmirovanie //Kiev: Nauk. dumka, 1991.- 236 s.
2. Lipaev V.V., Pozin B.A., Shtrik A.A. Tekhnologiia sborochnogo programmirovaniia. M.: Radio i sviaz, 1992.
3. Lavrischeva E.M., Grishchenko V.N. Sborochnoe programmirovanie. Osnovy industrii programmnykh produktov// Kiev: Nauk. Dumka, 2009 – 371 s.
4. Ershov A.P. Opyt integralnogo podkhoda k aktualnoi probleme PO. Kibernetika, 1984. S. 11-21.
5. Ershov A.P. Nauchnye osnovy dokazatelnogo programmirovaniia. Doklad AN SSSR, 1985. - s. 1–14.
6. Lavrishcheva E. M., Karpov L. E., Tomilin A. N. Podkhody k predstavleniiu nauchnykh znanii v Internet nauke. Sb. XIX Vserossiiskii nauchnoi konferentsii «Nauchnyi servis v seti Internet», Novorossiisk, 18-23 sentiabria 2017. - s. 310-326.
7. Lavrishcheva E. M., Karpov L. E., Tomilin A. N. Semanticheskie resursy dlia razrabotki ontologii nauchnoi i inzhenernoi predmetnykh oblastei// Nauchnyi servis v seti Internet: trudy XVIII Vserossiiskoi nauchnoi konferentsii (19-24 sentiabria 2016 g., g. Novorossiisk). — M.: IPM im. M.V. Keldysha, 2016. - s.126-138.- doi:10.20948/abrau-2016-8.
8. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation, 27 April 2007. https://www.w3.org/TR/SOAP12-part1
9. Web Services Description Language (WSDL) 1.1//W3C Note 15 March 2001. https://www.w3.org/TR/wsdl
10. Reference Model for Service Oriented Architecture 1.0. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html
11. Web Services Resource 1.2. WS-Resource - http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf, WS-Resource Properties - http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.
12. Lavrishcheva E.M. Teoria obiektno-komponentnogo modelirovania programmnyh sistem. Preprint ISP RAN, № 29, 2016 g. - M.: 48 s. ISBN 078-5-91474-025-9.
13. Lavrishcheva E.M. Komponentnaia teoriia i kollektsiia tekhnologii dlia razrabotki industrialnykh prilozhenii iz gotovykh resursov, Trudy 4-oi nauchno-prakticheskoi konferentsii «Aktualnye problemy sistemnoi i programmnoi inzhenerii», APSPI-2015, 20-21 maia 2015, s. 101-119.
14. K.M. Lavrischeva, A.K. Petrenko. Software Product Lines Modeling. Proceedings of the Institute for System Programming, vol. 28, issue 6, 2016, pp. 49-64. DOI: 10.15514/ISPRAS-2016-28(6)-4.
15. V.V. Kuliamin, E.M. Lavrischeva, V.S. Mutilin, A.K. Petrenko. Verification and analysis of variable operating systems. Proceedings of the Institute for System Programming, vol. 28, issue 3, 2016, pp. 189-208. DOI: 10.15514/ISPRAS-2016-28(3)-12.

16. Lavrishcheva E.M. Programmnaia inzheneria. Paradigmy, Tekhnologii, CASE-sredstva Programmirovania. 2 izd. – M.: Iurait, 2016. – 280 s.
17. Lavrishcheva E.M., Grishchenko V.N. Sviaz raznoiazykovykh modulei v OS ES. – M.: Finansy i statistika. 1982 - 136 s.
18. Lavrishcheva E.M. Programmnaia inzheneriia i tekhnologiia razrabotki programmnykh sistem. Iurait, 2016 - 431 s.
19. Ostrovskii A.I. Podkhod k obespecheniiu vzaimodeistviia programmnykh sred JAVA i MS.Net. - Problemy programmirovaniia, - 2011. №2. c. 37-44.
20. Theory and practice of software factories, K. M. Lavrischeva, 2011, Volume 47, Number 6, Pages 961-972.
21. Lavrishcheva E.M., Kolesnik A.L., Steniashin A.Iu. Obieektno-komponentnoe proektirovanie programmnykh sistem// Teoreticheskie i prikladnye voprosy.– Vesnik KNU, seriia fiz.-mat. nauk. – Kiev, 2013. – №4. – S. 150–164.
22. Ekaterina Lavrischeva, Andrey Stenyashin, Andrii Kolesnyk. Object-Component Development of Application and Systems. Theory and Practice /Journal of Software Engineering and Applications, 2014, http://www.scirp.org/journal/jsea.
23. Ekaterina M.Lavrischeva. Assemblling Paradigms of Programming in Software Engineering.- 2016, 9, 2016. - pp.296-317, http://www.scrip.org/journal/jsea, http://dx.do.org/10.4236/jsea.96021.
24. Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle, "Science and Information Conference-2015", Jule 28-30, London, UK, www.conference.thesai.org.- p.965-972.
25. Semantic Web programming//S. Hebeler, M. Fisher, R.Blace, A/Peter-Lopes, Willey Publiching Inc., 2008. http://semwebprogramming.org.
26. Semantic Web, Representation of data on the World Wide Web, based on the RDF standards. http://www.w3.org/2001/sw/
27. MacGregor S.D., Sykes D.A. Practical Guide to testing Object-oriented Software. Addison-Wesley Professional, 2001.
28. Lavrischeva E.M., Pakulin N.V., Ryzhov A.G., Zelenov S.V. Analysis of methods for assessing the reliability of equipment and systems. Practice of methods. Proceedings of the Institute for System Programming, vol. 30, issue 3, 2018, pp. 99-120. DOI: 10.15514/ISPRAS-2018-30(3)-8.
29. E.M. Lavrischeva, I.B. Petrov. Ways of Development of Computer Technologies to Perspective Nano, Future Technologies Conference (FTC-2017), 29-30 November 2017. Vancouver, Canada. - p. 539-547.