Моделирование прикладных и информационных систем из готовых сервисных ресурсов Интернет[1]

[1,2] *Лаврищева Е.М. <lavr@ispras.ru>*
[1] *Мутилин В.С. <mutilin@ispras.ru>*
[1,3] *Козин С.В. <kozyyy@yandex.ru>*
[1] *А.Г. Рыжов <ryzhov@ispras.ru>*

[1]*Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, Москва, ул. А. Солженицына, д. 25
[2]*Московский физико–технический институт,*
141700, Россия, Московская обл., г. Долгопрудный, Институтский пер., д. 9.
[3]*Национальный исследовательский университет Высшая школа экономики,*
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Рассматривается подход к созданию сложных (информационных и прикладных) систем из готовых ресурсов (модулей, компонентов, КПИ, сервисов, reuses и др.). В основе подхода создания систем, веб–систем лежит компонентная модель (КМ), включающая функциональные, системные, сервисные и интерфейсные готовые ресурсы (ГОР), и алгебра компонентов для выполнения разных операций над ГОР. Прикладные функции компонентов, КПИ описываются в языках программирования (ЯП), интерфейсы в языках IDL и WSDL, а сервисные компоненты создаются в SOA, SCA IBMSphere или выбираются из Интернет библиотек, как готовые. Исходные компоненты верифицируются и сохранятся в репозитории ГОР и интефейсов. Представлен метод сборки ГОР: Link в среде IBM, MS.VS; make в BSD, config в JavaEE; стандарт IEEE 828–96–2012 (Configuration), как итог всех сборок. ГОР тестируются на множестве тестовых данных, проверяется их правильность и надежность работы. На готовый сконфигурированный продукт формируется сертификат качества на основе стандартов ISO/IEC 9000 (1–4) quality. Даны перспективы развития средств обеспечения безопасности и качества веб–систем.

Ключевые слова: компонент; сервис; ресурс; система; веб–система; сборка; надежность; качество.

DOI: 10.15514/ISPRAS–2019–31(1)–1

Для цитирования: Лаврищева Е.М., Мутилин В.С., Козин С.В., Рыжов А.В. Моделирование прикладных и информационных систем из готовых сервисных ресурсов Интернет. Труды ИСП РАН, том 31, вып. 1, 2019 г., стр. 7–24. DOI: 10.15514/ISPRAS–2019–31(1)–1

1. Введение

В 70–80 годы прошлого столетия сформировался подход к сборке разнородных модулей в сложную систему с использованием интерфейсов при реализации комплексов программ специального и общего назначения в рамках ВПК (под руководством Липаева В.В.) и он получил название – сборочного программирования [1–3]. Метод сборки был представлен в ОС ЕС (IBM–360) и в других средах – Sun Microsystems, MS VS .Net, IBM WebSphere и др. По словам А.П. Ершова [4], «сборочное программирование обеспечивает построение уже существующих (проверенных на правильность) готовых отдельных фрагментов программ (типа reuses) в сложную структуру». Для описания интерфейсов использовался простой язык

[1] Работа выполняется по проекту РФФИ № 16-01-00352

MIL, затем после 90–х появились стандартные языки IDL, WSDL и др., а также средства сборки link и make BSD (1996), .Net, Java, SPAROL и др. [5–10]. Оператор make обеспечивают сборку исполняемых модулей из Filemake библиотек и задает им порядок связей друг с другом в средах ОС Linux, Java, .NET и др. Метод сборки получил широкое развитие на фабриках программ (Д. Гринфилда (Jack Greenfield) и Г. Ленц (Herbert Lenz) – потоковая сборка, 2007; К. Чернетски (Krzysztof Czarnecki) и А. Айзенекер (Ulrich Eisenecker) – мультисборка, 2005; И. Бей (Ying Bai) – взаимодействие разноязыковых программ в разных средах, 2010; К. Пол (Klaus Pohl) – конвейерная сборка на ProductLine/ProductFamily, 2004 и др.). Создана теория фабрик для производства программ из КПИ [20–23], основанная на методе сборки (на статью [20] приходят запросы из Китая, Таиланда, Южной Кореи и др.).

В 2009 году (потом в 2012) процесс сборки (интеграции) готовых reuses был стандартизирован (IEEE 828–96–2012, Configuration) и позволяет получить конфигурационную структуру любой программной и информационной системы, элементы которой можно менять, добавлять, заменять и удалять. Затем делать новый вариант конфигурационной структуры системы.

2. Подход к моделированию систем из компонентов

Моделирование систем проводится с помощью компонентной модели (КМ), как составной модели ОКМ [11]. Компонент (Component, Comp, C) определяется как некоторая абстракция, включающая в себя интерфейсный раздел и артефакт описания функции. Он может иметь несколько реализаций в зависимости от операционной среды, модели данных, СУБД и др. [12]. Интерфейс определяет данные для связи одного компонента с другими. Компонент может иметь несколько интерфейсов и наследуется в виде классов в модели компонента или каркаса на любом ЯП (C, C++, Ruby, Basic, Java и др.). Компоненты могут быть:

1) программные, сервисные, системные и служебные;
2) серверные, клиентские и веб–серверные и веб–клиентские;
3) контейерные, патерные, программные, математически–ориентированные и др.

Каждый из этих типов компонентов имеет спецификацию, требования, правила их взаимодействия с другими компонентами, заданную в интерфейсе. Для компонентов разработана компонентная алгебра (внешняя, внутренняя и эволюционная), которая включает разные действия над компонентами (добавление, удаление, замена, верификация, тестирование и др.).

2.1 Алгебра операций над компонентами

Операция добавления компонента C к компонентой среде обозначается $\oplus$ и выполняется согласно правил

$C \oplus CE_1 = CE_2$,

$NameSpace = \{C.CName\} \cup CE_1.NameSpace$,

$CE_2.InRep = \{C.(CIn_i, CName)\} \cup CE_1.InRep$,

$CE_2.ImRep = \{C.(C_j, CName)\} \cup CE_1.ImRep\}$.

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 знаком $\Diamond$ и имеет вид: $CE_1 \Diamond C = CE_2$.

Теорема 2. Для любого компонента C в среде CE выполняется равенство

$(C_2 \oplus CE) \Diamond C = CE$.

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

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

Операция объединения (∪) 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 интерфейса и реализации компонентов
Операция добавления *addImp Comp* нового входного интерфейса и реализации компонента Comp в среду или в модель:
NewComp = AddImp(OldComp, NewCImps, NewCIntOs) и
NewInt = OldInt ∪ NewIntOs, где
NewCImp = OldCImp ∪ {*NewImps*}(∃ *OldIntt* ∈ *OldCIntI*) *Provide*(*OldInt*) ⊆ *NewImps*. где
NewCImps – новая реализация, которая добавляется;
OldCInt – множество существующих интерфейсов *Int*.
Условие целостности компонента выполняется автоматически при условии использования прежних входных интерфейсов. Из целостности старого компонента вытекает целостность нового компонента.

2.2 Компонентная алгебра

Компонентная алгебра – это внешняя, внутренняя и эволюционная алгебры**:**

$\sum = \{\varphi_1, \varphi_2, \varphi_3\}$, 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 $\varphi_1 = \{CSet, CESet, \Omega_1\}$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:
- add imp 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;
- переименования компонентов и интерфейсов и добавления новых компонентов в ПС;
- реверсной инженерии – восстановления исходной структуры компонента по выходному коду ПС.

Приведенные алгебры позволяют гибко управлять компонентами в моделируемой системе.

3. Проектирование и верификация моделей систем

Построение архитектуры системы на компонентной основе проводится с помощью графовой модели метода ОКМ, создаваемой с помощью логико–математического аппарата на четырех уровнях:
- обобщающий уровень, который задает объект в виде денотата в соответствии с теории Фреге типа <имя объекта> <концепт>;
- структурный уровень для теоретико–множественного упорядочения объектов и представления их в виде графа; могут применяться операции объединения, пересечения, разности и т.п.;
- характеристический уровень для логико–алгебраического определения характеристик объектов и их свойств;
- поведенческий уровень для определения поведения и взаимосвязей между объектами графа ОМ.

3.1 Графовая модель

На первых двух уровнях выделяются объекты $O = (O_i, O_2, \ldots, O_n)$ или функции предметной области (ПрО) и создается граф G, в вершинах которого размещаются O_i, а дуги задают их связи (рис.1).

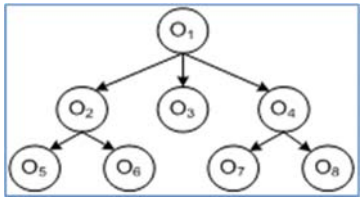

Рис.1. Граф из объектов ПрО
Fig. 1. Graph of domain objects

Свойства графа G:
- для каждой вершины Oi существует хотя бы одна связь (структурная) с другой вершиной графа (стрелки);
- существует лишь одна вершина O1 графа G, которая имеет статус множества объектов, отображающего ПрО в целом.

Графу G соответствует объектная модель $OM = (O_1, O_2, ..O_7)$. Объекты модели проверяются с помощью model checking на соответствие спецификации требованиям и свойствам (рис.2).

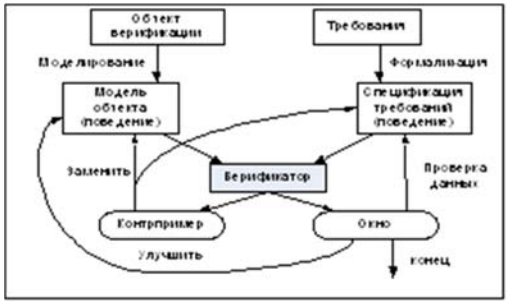

Рис.2. Верификация модели ОМ
Fig.2. Object model verification

Если соответствие удовлетворяется, то верификатор сообщает о правильности модели, в противном случае дается пояснение о возникшем несоответствии.

На следующих уровнях моделирования системы уточняется функции и интерфейсы. В результате в граф G добавляются интерфейсов связи объектов друг с другом $\boldsymbol{I}_{Oi} = \{Io_{n5}, I_{o6}, I_{o7}, I_{o8}\}$ (рис.3.).

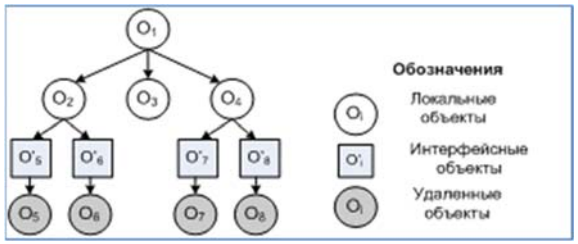

Рис. 3. Граф G с объектами и интерфейсами
Fig. 3. Graph G with objects and interfaces

В графе *G* заданы:

- O1, O2, O3, O4 , O5 , O6 , O7 , O8 – функциональные объекты;
- O′5, O′6, O′7, O′8, – интерфейсные объекты, которые размещаются в репозитории, а дуги соответствуют связям между всеми видами объектов.

Элементы графа $O_1 – O_8$ описываются в ЯП (Fortran, Basic, C, C++, Python и др.), а интерфейсные объекты $O'_5 – O'_8$ в языке IDL/ WSDL. По графу G можно построить программы $P_0 – P_5$ с использованием операторов сборки *link* :

$P_0 = (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5)$.

$P_1 = O_2 \cup O_5$, *link* $P_1 = In\ O'_5\ (O_2 \cup O_5)$:

$P_2 = O_2 \cup O_6$, *link* $P_2 = In\ O'_6\ (O_2 \cup O_8)$:

P_3:

$P_4 = O_4 \cup O_7$, *link* $P_4 = In\ O'_7\ (O_4 \cup O_7)$:

$P_5 = O_4 \cup O_8$, *link* $P_4 = In\ O'_8\ (O_4 \cup O_8)$:

На основе этого графа определяется модель системы (sys) в виде:

- M_{sys} = < Mf, Ms, Mi, Md>, где
- M_f = (fO1, fO2,…, fOr) – множество функций объектов ПрО;
- M_s = (Sin, Sout, Sinout) – множество входных данных Sin, выходных данных Sout и промежуточных данных Sinoutt, определяемых на множестве системных сервисов (Common Facility Services) операционной среды;
- M_i = (Io1, Io2, …, Ion) – множество интерфейсно–компонентных элементов для функциональных элементов fOn входные in, выходные параметры out и inout из множества M_s;
- M_d = (Md1, Md2, …, Mdn) множество данных и метаданных ПрО, с которыми работает система.

Множества M_{sys}, M_s и M_i определяют ГОР, интерфейсы и общие данные. Они могут иметь пересечение по данным, которые относятся к *in*, *out*, *inout* и входят в состав внешних типов данных, которыми обмениваются элементы модели через сервер Application.

Модели проверяются и на их основе конфигурируются отдельные версии вариантов выходной системы. Для каждой пары отдельных функциональных и интерфейсных элементов может быть задана точка их изменения для замены более корректным или функционально правильным компонентом.

Системные компоненты управляют оборудованием, ОС и серверами. В состав системы входят клиент, сервер или Интернет браузер. На веб–сервере происходит обработка запросов на выполнение разных операций над программами и данными.

3.1 Модели системных и прикладных сервисов клиент–серверной архитектуры

Основу сервисов составляет клиент–серверная архитектура Интернет, которая первоначально была реализована в CORBA [13, 16, 17] и включает:

- брокер объектных запросов (*Object Request Broker* – ORB) для взаимодействия 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.

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

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;
- интерфейса динамического вызова (Dynamic Invocation Interface – DII) объекта, определяемого во время выполнения программы клиента при поиске интерфейса в репозитории интерфейсов или в репозитории реализаций;
- интерфейса сервисов ORB (ORB Services Interface), содержащего набор сервисных функций, которые клиент запрашивает у сервера через брокер ORB.

Stub–интерфейс обеспечивает взаимосвязь клиента с ORB через *stub* и посылает параметры серверу в запросе. Схема взаимодействия клиента с объектом соответствует схеме вызова RPC удаленных процедур.

Интефейс DII обеспечивает доступ объектов и их интерфейсов во время выполнения. В каждом вызове указывается тип объекта, тип запроса и параметры. По этой информации извлекается соответствующая программа из репозитория интерфейсов и репозитория реализаций [18].

Описание интерфейса в IDL начинается с ключевого слова **interface**, за которым следует: имя интерфейса, описание типов параметров и операций вызова объектов. Описание типов данных начинается ключевым словом typedef, за которым следует базовый или конструируемый тип и его идентификатор. В качестве константы может быть некоторое значение типа данного или выражение, составленное из констант. Типы данных и константы описываются как фундаментальные типы данных ЯП: integer, boolean, string, float, char и др.

3.2.1 Генерация системных программ клиента и сервера

Генерируются два класса системных компонентов: клиент; сервер.

В клиент помещаются механизмы коммуникации между программными компонентами. Пользователь этого класса не должен ничего менять, а использовать 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, отвечающие задекларированным методам, описанным в интерфейсе. Для использования сервера пользователю необходимо реализовать конкретную логику этих методов следующим образом:

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

3.2.2 Современная клиент–серверная архитектура в Интернет

В связи с большим количеством пользователей Интернет и большими объемами данных (типа Big Data), современная клиент–серверная архитектура включает больше серверов, в том числе и серверов Баз Данных. Они способствуют увеличению общей производительности веб–систем путем вертикального и горизонтального масштабирования.

Вертикальное масштабирование увеличивает производительность за счет добавления аппаратных ресурсов серверов, а горизонтальное масштабирование способствует росту производительности систем за счет увеличения количества серверов и БД. Это масштабирование основываться на процессах: *Front–end* для клиентских систем и *Back–end* для обслуживания серверных частей приложений. Эти процессы должны обеспечивать безопасность, защиту и качественную работу в Интернет [28].

3.3 Средства тестирования систем

Тестирование системы, созданной методом сборки, проводится с помощью набора тестов для отдельных элементов и системы в целом. Тесты проверяют функциональные и интерфейсные компоненты. В качестве инструмента тестирования можно использовать фреймворк Visual Studio 2007 со средствами проверки правильности тестирования разных видов объектов. В него входит компонент *Test Manager*, который управляет средствами планирования процесса тестирования и выполнения тестовых сценариев. При обнаружении ошибок в процессе тестирования вносятся исправления в модели системы M_{sys}. Затем проводится повторная операция config для получения готового продукта с добавлением новых объектов или удаления старых.

4. Проектирование систем из сервисных компонентов

В настоящее время в Интернет накоплено огромное количество ГОР сервисно–компонентного типа, которые представляются средствами моделей SOA (Service Oriented Architecture) и SCA (Service Component Architecture). Интерфейс сервисных компонентов задается в языке WSDL [13 –15]. Элементы SOA задают описание некоторой функции (Function) с заданным качеством сервиса (Quality service) в IT–стандартах комитета W3C. Средствами модели SCA задаются сервисные компоненты, как ГОР типа reusable. К ним

относится EJB сервер приложений J2EE, сетевые сервисы, объекты планирования, доступа к БД и к системе. Элементы архитектуры SCA могут собираться в систему путем интеграции или конфигурационной сборки. Данные модели SOA и SCA используются нами при моделировании систем и веб–систем. При проектировании веб—систем создаются:

- модель системы M_{wsys};
- клиент–серверная архитектура с веб–сервером и веб–клиентом для выполнения запросов от M_{wsys};
- спецификации интерфейсов компонентов I_{on};
- схемы запросов к имени функции или компонента с помощью операторов вызова или протоколов связи стандарта ISO/IEC;
- сервисно–компонентные элементы для сред (J2EE, .Net, Appach, JAVA, SOAP, WSDL и др.) [15] и их накопление в библиотеках системных сред Microsoft VS .Net,, IBM, Intel, Linux. Semantic Web Интернет;
- данные для верификации и тестирования сервисных компонентов.

4.1 Создание сервисных компонентов в SOA

Модель SOA – это набор принципов и средств создания системного ПО и прикладных программных систем с помощью совместимых и унифицированных сервисов Интернет [13–15]. SOA задает реализацию сервисов на серверной стороне с открытым интерфейсом 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. 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.).

Фундаментом сетевых служб SOA являются:

- 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.);
- поставщик услуги, который осуществляет ее реализацию в виде сетевой службы, прием и выполнение запросов пользователей, а также публикацию сведений о сервисе в соответствующем реестре;
- реестр (каталог) служб содержит библиотеку сервисов, а также средства их поиска и вызова с помощью запросов, которые поступают от поставщиков сервисов на получение сервисов.

Клиент осуществляет поиск и вызов необходимого сервиса из реестра описания сервисов в соответствии с заданным интерфейсом.

Посредником между этими сервисами и системой является *провайдер* клиент–серверной архитектуры, который обеспечивает взаимодействие между поставщиками и провайдерами.

4.2 Создание сервисно–компонентных ГОР в SCA (IBM WebSphere)

SCA [21] предназначена для работы с прикладными компонентами с разными спецификациями 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 в IBM WebSphere Integration Developer (WebSphereID) упаковываются 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 и передавать данные для обработки и интеграции. Система

Dynamic Profiles WebSphere Portlet Factory обеспечивает динамическую конфигурацию пользовательского интерфейса, 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 (СПП, [18]). В ней каждый программный элемент содержит удаленные компоненты типа reuses – КПИ, которые обмениваются гетерогенными данными и обеспечивают выполнение системы. При этом используются механизмы сервисных объектов данных SDO и сервисы доступа DAS.

Для описания сетевых сервисов язык WSDL предоставляет 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 взаимодействия клиента и сервира. 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.

4.3 Средства Семантик Веб для разработки ГОР и систем

Для описания ГОР имеются такие средства [24, 25]:

- **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 [23]),
  - **WSCL** (Web Services Conversation Language [24]),
  - **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.

4.4 Сборка сервисных ГОР веб–системы в Интернет

Для сборки сервисов используется инструмент – ***Jopera for Eclipse*** (http://www.jopera.ethz.ch/), который обеспечивает [18, 23– 25]:

- 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 Web–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 Web–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 систему 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.

4.5 Конфигурация ресурсов веб–систем

Под *конфигурацией* системы понимается структура некоторой ее версии, включающая функции, объединенные между собой операциями связи с параметрами, задающими режимы

функционирования системы [16–18]. Версия или конфигурация системы согласно IEEE Standard 828–2012 (Configuration) включает:

- базис конфигурации – BC (Configuration Baseline);
- элементы конфигурации (Configuration Item);
- компоненты, ГОР, входящие в описание моделей Msys, Mwsys;

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 исходя из стандарта состоит в выполнении следующих задач:

1) идентификация конфигурации (Configuration Identification).
2) контроль конфигурации (Configuration Control).
3) учет статуса конфигурации (Configuration Status Accounting).
4) фудит конфигурации аудит (Configuration Audit).
5) трассировка изменений конфигурации на этапах сопровождения и эксплуатации системы;
6) верификация компонентных сервисов по моделями M_{sys}, M_{wsys};
7) доказательство изоморфного отображения данных компонентных сервисов с типами данных стандарта ISO/IEC 11404 Object Data Types –2007.

При конфигурационной сборке ГОР используется модели систем M_{sys}, M_{wsys} и модель характеристик MF (Model Feature). ГОР и КПИ накапливаются в репозиториях или библиотеках системы. Они отбираются, адаптируются и интегрируются в единую систему. Основную роль в этих процессах выполняет конфигуратор ИТК (http://7dragons.ru/ru). Он обеспечивает сборку разнородных ГОР и их интерфейсов с вариантами отдельных готовых продуктов, которые находятся в репозиториях.

М*одель среды конфигуратора* включает:

- описание графовой схемы системы из ГОР;
- модели вариантов системы;
- конфигурационную модель ГОР и КПИ;
- операцию конфигурационной сборки КПИ и ГОР;
- аудит конфигурации;
- верификатор моделей и ГОР.
- оценку качества ГОР.

Конфигуратор собирает требуемые ГОР и КПИ по моделям в веб–систему с учетом их интерфейсов по заданным моделям и формирует конфигурационный файл системы, который выполняется в соответствующей операционной среде.

5. Моделирование варианта ОС для прикладных систем

Выше рассмотрены подходы к созданию систем и веб–систем из ГОР, сделанных другими разработчиками для выполнения разных функций математического, системного и сервисного типа. В данном разделе рассматриваются подход к процессам создания варианта ОС Linux, который будет управлять некоторой новой прикладной областью (например, медицина, биология, геология и др.). Разработан процесс создания варианта ОС в виде экспериментального варианта ядра ОС [19, 30–33]. Ниже дается описание.

5.1 Процесс определения варианта ядра ОС Linux

Для определения варианта ОС необходимо выполнить 7 процессов следующего содержания [19, 30–33].

1) **Конфигурирование.** Во время сборки Linux проверяет файл конфигурации kconfig на наличие рекурсивных зависимостей и несуществующих переменных в коде

2) **Поиск «мертвого кода»,** т.е. такого кода, в котором контроль не передается ни при каких обстоятельствах.
3) **Препроцессирование** включает предварительную обработку элемента функции из ядра ОС непосредственно перед компиляцией и получения кода для окончательной версии программы. Если находятся ошибки, то выдается информация о них.
4) **Компиляция** состоит в выдаче кода или случайных ошибок при обнаружении необъявленной переменной / функции, отсутствие пунктуации в коде и т. д.
5) **Связывание** отдельных элементов ядра выполняет оператор Link. Он находит ошибки задания связи компонентов в интерфейсе или ошибки вызова компонентов из внешних библиотек.
6) **Обработка** ошибочных ситуаций.
7) **Тестирование** собранного варианта ОС. Обработка ошибок проводится с помощью LDV и CPAchecker. Инструмент LDV ставит метки кода в соответствии с заданными правилами, CPAchecker проверяет доступность меток и правильность выполнимости компонентов.

5.2 Конфигурация варианта ядра ОС Linux

На основе анализа ядра ОС Linux установлено, что оно содержит более 10 000 переменных и большое количество функциональных и системных компонентов для обработки разного рода заданий по функционированию любых прикладных систем. Создание некоторого варианта ОС для класса прикладных систем (например, для медицины, биологии и др.) требует выбора из множества компонентов ОС наиболее подходящих для оперативного управления прикладными системами. Из выбранных компонентов ОС создается модель *MF* с базовыми характеристиками компонентов ОС и модель системы *Msys* варианта ОС, включающая множество функциональных M_{fi}, интерфейсных M_{io} и компонентов M_d работы с данными. Эти компоненты тестируются на правильность их идентификации с помощью наборов тестов и операций установления связей с соответствующими компонентами других множеств. После тестирования проводится операция *config* (M_{fi}, MI_{oi}, M_{di}) для получения конфигурационного файла варианта ОС.

Процесс формирования варианта ОС включает [4–8] определение загружаемых модулей из библиотек сервисов, драйверов устройств и файловых систем и настройки параметров безопасности, защиты и криптографии.

Конфигурация ядра варианта ОС проводится с помощью этих параметров с учетом особенностей прикладной системы. Например, ядро может включать в себя множество опций безопасности исходя из стека SELinux Национального агентства по безопасности NSA, ориентированных на безопасность функциональных компонентов ОС.

Перед конфигурацией варианта системы ОС проверяется файл /etc /udev /rules.d/70–persistent–net.rules и определяются имена сетевых устройств, а также схема именования правилами Udev. Выходной файла начинается с блока комментариев, за которым следуют две строки для каждого сетевого адаптера. Первая строка сетевой карты – это описание и комментарии, показывающие идентификаторы оборудования и устройств согласно карты PCI и драйверов.

При задании имени интерфейса идентификатор аппаратного обеспечения не используются. Вторая строка – это правило Udev, которое соответствует сетевой карте с фактически присвоенным именем.

Некоторое программы ПО ОС могут устанавливаться позже с помощью символических ссылок /dev /cdrom и /dev/dvd для устройства CD–ROM или DVD–ROM. Кроме того, могут

помещаться символические ссылки в /etc/fstab. Udev в зависимости от возможностей каждого устройства.

Сценарий может работать в режиме «by–path» по умолчанию для устройств USB и FireWire, создаваемые правила которых зависят от физического пути к устройству CD или DVD. Сценарий может работать в режиме «по–id» (по умолчанию для устройств IDE и SCSI) и зависит от правил идентификационных строк, хранящихся на устройстве CD или DVD. Физический путь к устройству (порты и / или слоты) изменяются, например, при планировании перемещения диска в другой порт IDE или другой разъем USB режима «by–id».

Для каждого устройства находится соответствующий каталог в разделе / sys / class или / sys / block и для видеоустройств в разделе / sys / class / video4linux / videoX.

Интерфейсы сетевых сценариев задаются в файле / etc / sysconfig /. При этом файл ifconfig содержит настраиваемый интерфейс ifconfig.xyz в сетевой карте с именем eth0. Внутри этого файла содержатся атрибуты интерфейса IP–адрес, маски подсети и т. д. (Пример описания в сконфигурированного файла см. в [28].)

6. Обеспечение качества систем

Качество – это совокупность свойств (показателей качества) ПО, которые обеспечивают его способность удовлетворять потребности заказчика, в соответствии с его назначением. Оно задано стандарте ГОСТ 2844–98 с помощью модели качества и его показателей. Стандарт ISO/IEC 12207 определил основные процессы ЖЦ разработки ПС, но и организационные и дополнительные процессы, которые регламентируют планирование, управление качеством и оценку затрат на проект. На этапах ЖЦ проводится анализ качества ПО, ориентированные на:

- 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 отказах, дефектах и др. ошибках в системе и оценивание надежности по соответствующим моделям надежности.

Модель качества ПО согласно стандарту задает шесть показателей (характеристик) 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).

Каждая характеристика q_i рассчитывается по специальным формулам и метрикам стандарта. Надежность оценивается согласно полученных на процессе тестирования ошибок, дефектов и отказов в ПО и по разным моделям надежности (оценочным, измерительным и др.).

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

$$q_i = \sum_{j=1}^{k_i} a_{i_j} m_{i_j} w_{i_j}$$

где a_{i_j} – атрибуты каждого показателя качества; m_{i_j} – метрики каждого атрибута качества; w_{i_j} – вес каждого атрибута показателя качества системы. Полученные данные по характеристикам (показателям) модели качества входят в сертификат качества.

7. Заключение

Данный подход к сборке систем и веб–систем реализован в рамках проекта РФФИ №16–01–00352 «Теория и методы разработки изменяемых программных систем» [14]. В работе рассмотрены базовые понятия – ГОР, модели систем и конфигурационной сборки. Они специфицируются в языках (C++, JAVA, Python, Basic и др.). Описан компонентный метод ОКМ, основу которого составляет логико–математический аппарат проектирования и формирования модели системы и модели характеристик MF. Сформулированы модели характеристик масштабированных клиент–серверных архитектур при создании современных веб–систем [28]. Приведено описание новых открытых моделей SOA и SCA для описания сервисов, серверов и клиентов. Дано описание разных методов сборки систем из ГОР (link, make, config и др.). Приведен вариант веб–системы для прикладных систем. Обоснованы перспективы развития технологии программирования прикладных систем с обеспечением безопасности, надежности и качества в клиент–серверной архитектуре [30–33].

Список литературы

[1] Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Киев, Наук. думка, 1991, 236 стр.

[2] Липаев В.В., Позин Б.А., Штрик А.А. Технология сборочного программирования М.: Радио и связь, 1992, 324 стр.

[3] Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. Киев, Наук. думка, 2009, 371 стр.

[4] Ершов А.П. Опыт интегрального подхода к актуальной проблеме ПО. Кибернетика, № 3, 1984, стр.11–21.

[5] Лаврищева Е. М., Карпов Л. Е., Томилин А. Н. Подходы к представлению научных знаний в Интернет науке. Сб. трудов XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18–23 сентября 2017, стр. 310–326.

[6] Лаврищева Е. М., Карпов Л. Е., Томилин А. Н. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей. Сб. трудов XVIII Всероссийской научной конференции «Научный сервис в сети Интернет» Новороссийск, 19–24 сентября 2016 г., стр. 126–138.

[7] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation 27, April 2007, URL: http://www.w3.org/TR/SOAP12–part1.

[8] Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001, URL: http://www.w3.org/TR/wsdl.

[9] Reference Model for Service Oriented Architecture 1.0. URL: http://docs.oasis– open.org/soa–rm/v1.0/soa–rm.html

[10] Web Services Resource 1.2 (WS–Resource). URL: http://docs.oasis– open.org/wsrf/wsrf–ws_resource–1.2–spec–os.pdf.

[11] Лаврищева Е.М. Теория объектно–компонентного моделирования программных систем. Препринт 29 ИСП РАН, 2016, 48 стр. ISBN 078–5–91474–025–9.

[12] Лаврищева Е.М. Компонентная теория и коллекция технологий для разработки индустриальных приложений из готовых ресурсов. Труды научно–практической конференции «Актуальные проблемы системной и программной инженерии», АПСПИ–2015, 20–21мая 2015, стр. 101–119.

[13] Лаврищева Е.М. Программная инженерия. Тема 1. Теория программирования, 50 стр.; Тема 2. Технология программирования, 48 стр.; Тема 3. Базовые основы программной инженерии, 52 стр. Методические пособия, Москва, МФТИ, 2016.

[14] Лаврищева Е.М., Петренко А.К. Моделирование семейств программных систем. Труды ИСП РАН, том 28. вып. 6, 2016 г,, стр. 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 ИСП РАН, том 28, вып.3, 2016 г., стр. 189–209. 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 стр.

[17] Лавришева Е.М., Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС. М., Финансы и статистика, 1982, 136 стр.
[18] Лаврищева Е.М. Программная инженерия и технология разработки программных систем. М., Юрайт, 2017, 431 стр.
[19] Островский А.И. Подход к обеспечению взаимодействия программных сред JAVA и MS.Net. Проблемы программирования, № 2, 2011 г., стр. 37–44.
[20] Лаврищева Е.М. Теория и практика фабрик программ. Кибернетика и системный анализ, том 47, no. 6, 2011 г., стр. 145–158.
[21] Лавріщева К.М., Колесник А.Л., Стеняшин А.Ю. Об'єктне-компонентне проектування(ОКП) програмних систем. Теоретичні і практичні питання. Вісник КНУ, серия физ.–мат. наук, спецвипуск, 2013 г., стр. 150–164 (на украинском).
[22] Ekaterina M. Lavrischeva. Assemblling Paradigms of Programming in Software Engineering. Journal of Software Engineering and Applications, vol. 9, no. 6, 2016, pp. 296–317, DOI: 10.4236/jsea.2016.96021.
[23] Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle. In Proc. of the 2015 Science and Information Conference (SAI), Jule 28–30, London, pp. 965–972.
[24] John Hebeler, Matthew Fisher, Ryan Blace, Andrew Perez-Lopez. Semantic Web programming. Willey, 2009, 652 p.
[25] Semantic Web. Representation of data on the World Wide Web based on the RDF standards. URL: http://www.w3.org/2001/sw/.
[26] John D. McGregor, David A. Sykes. Practical Guide to testing of Object–oriented software. Addison–Wesley Professional, 2001, 416 p.
[27] Sayyad A. S., Ingram J., Menzies T., Ammar H. Scalable product line configuration: a straw to break the camel's back. In Proc. of the IEEE/ACM 28th International Conference on Automated Software Engineering (ASE 2013), 2013, pp. 465–474.
[28] Лаврищева Е.М., Рыжов А.В. Подход к моделированию систем и сайтов из готовых ресурсов. В сб. трудов XX Всероссийской научной конференции «Научный сервис в сети Интернет», Новороссийск, 17–22 сентября 2018 г. CEUR Workshop Proceedings, vol. 2260, pp. 321–345.
[29] Козин С.В. Конфигурационная сборка варианта ядра Linux для прикладных систем. Труды ИСП РАН, том 29, вып.3, 2018, стр. 161-170. DOI: 10.15514/ISPRAS-2018-30(6)-9.
[30] E.M. Lavrischeva, A.K. Petrenko. Informatics: Formation of computer software and technologies of software systems. ISP RAN/Proc. ISP RAS, 2018, vol. 30, issue 5, pp. 7-30. DOI: 10.15514/ISPRAS-2018-30(5)-1.
[31] Лаврищева Е.М. Информатика и ЭВМ-70. Анализ и аспекты развития. Доклад на Открытой конференции ИСП РАН им. В.П. Иванникова, 2018.
[32] E. M. Lavrischeva. The Scientific Basis of Software Engineering. International Journal of Applied and Natural Sciences (IJANS), vol. 7, issue 5, 2018, pp. 15–32.
[33] 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. 4., 2018, стр. 99-120. DOI: 10.15514/ISPRAS-2018-30(3)-8.

Modeling of application and information systems from ready–made Internet service resources

[1,2] *Lavrischeva E.M. <lavr@ispras.ru>*
[1] *Mutilin V.S. <mutilin@ispras.ru>*
[1,3] *Kozin S. V. <kozyyy@yandex.ru>*
[1] *Ryzhov A.V. <ryzhov@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Moscow Institute of physics and technology, 9, Institutsky per., Dolgoprudny, Moscow region, 141700, Russia*
[3] *National research University Higher School of Economics, 20, Myasnitskaya st., Moscow, 101000, Russia*

Annotation. The approach to creation of complex (information and applied) systems from ready resources (modules, components, reusable components, services, reuses, etc.) is considered. At the heart of the approach of creating systems, Web systems is a component model (CM), which includes functional, system, service and interface resources (GOR), and the algebra of components to perform various operations on the GOR. Application functions components of the KPI are described in the programming languages (PL), interfaces in languages IDL and WSDL, and service components are created in SOA, SCA IBM Sphere, or get out of the Internet libraries, as a ready. The initial components are verified and stored in the repository of GOR and interfaces. The method of assembling GOR: Link in IBM and MS.VS environment are presented; make in BSD, config in JavaEE; IEEE standard 828–96–2012 (Configuration). GOR tested on many test data, verified their accuracy and reliability. A quality certificate based by ISO/IEC 9000 (1–4) is generated for the finished configured product. The prospects of development of security and quality of web systems are given.

Keywords: component; service; resource; system; web–system; Assembly; reliability; quality.

DOI: 10.15514/ISPRAS-2019-31(1)-1

For citation: Lavrischeva E.M., Mutilin V.S., Kozin S.V., Ryzhov A.V. Modeling of application and information systems from ready–made Internet service resources. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 1, 2019. pp. 7-24 (in Russian). DOI: 10.15514/ISPRAS-2019-31(1)-1

References

[1] Lavrischeva E.M., Grishchenko V.N. Assembly programming. Kyiv, Nauk. Dumka, 1991, 236 p. (in Russian).
[2] Lipaev V.V., Posin B.A., Strick A.A. Technology of Assembly programming. M., Radio and communication, 1992, 324 p. (in Russian).
[3] Lavrischeva E. M., Grishchenko V. N. Assembly programming. The basics of the software product industry. Kiev, Nauk. Dumka, 2009, 371 p. (in Russian).
[4] Ershov A.P. Experience of integral approach to the actual problem of Software. Cybernetics, № 3, 1984, pp. 11–21 (in Russian).
[5] Lavrischeva E.M., Karpov L.E., Tomilin A.N. Approaches to the representation of scientific knowledge in Internet science. In Proc. of the XIX All–Russian scientific conference "Scientific service on the Internet", Novorossiysk, 18–23 September 2017, pp. 310–326 (in Russian).
[6] Lavrischeva E.M., Karpov L.E., Tomilin A.N. Semantic resources for development of ontology of scientific and engineering subject areas. In Proc. of the XVIII All–Russian scientific conference "Scientific service on the Internet", Novorossiysk, September 19–24, 2016, pp. 126–138 (in Russian).
[7] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation 27, April 2007, URL: http://www.w3.org/TR/SOAP12–part1.
[8] Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001, URL: http://www.w3.org/TR/wsdl.
[9] Reference Model for Service Oriented Architecture 1.0. URL: http://docs.oasis– open.org/soa–rm/v1.0/soa–rm.html.

[10] Web Services Resource 1.2 (WS–Resource). URL: http://docs.oasis– open.org/wsrf/wsrf–ws_resource–1.2–spec–os.pdf.
[11] Lavrischeva E.M. Theory of object–component modeling software systems. Preprint 29, ISP RAS, 2016, 48 p., ISBN 078–5–91474– 025–9 (in Russian).
[12] Lavrischeva E.M. Component theory and collection of technologies to develop industrial applications from ready–made resources. In Proc. of the 4th scientific–practical conference "Actual problems systems and software engineering", APSSE–2015, 20–21 may 2015, pp. 101–119 (in Russian).
[13] Lavrischeva E.M. Software engineering. Topic 1.Theory Programming, 50 p.; Topic 2. Programming technology, 48 p.; Topic 3. Fundamentals of software engineering, 52 p. Methodical manuals, Moscow, MIPT, 2016 (in Russian).
[14] Lavrischeva E.M., Petrenko A.K. Software Product Lines Modeling. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 6, 2016, pp. 40-64. DOI: 10.15514/ISPRAS-2016-28(6)-4 (in Russian).
[15] Kulyamin, V.V., Lavrischeva E.M., Mutilin V.S., Petrenko A.K. Verification, and analysis of various operating systems. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 3, pp. 189–208. DOI: 10.15514/ISPRAS-2016-28(3)-12 (in Russian).
[16] Lavrischeva E.M. Software engineering. Paradigms, Technologies, CASE-Software. 2nd ed. Moscow, Yurayt, 2016, 280 p. (in Russian).
[17] Lavrischeva E.M., Grishchenko V.N. Communication of multilingual modules in the EU OS. M.: Finance and statistics, 1982, 136 p. (in Russian)
[18] Lavrischeva E.M. Software engineering and technology for development of software systems. Moscow, Yurayt, 2017, 431 p. (in Russian)
[19] Ostrovsky A.I. Approach to software interaction JAVA environments and MS .Net. The problems of programming, №2, 2011, pp. 37–44 (in Russian).
[20] Lavrischeva E. M. Theory and practice of software factories. Cybernetics and Systems Analysis, volume 47, issue 6, November 2011, pp 961–972. DOI: 10.1007/s10559-011-9376-5.
[21] Lavrischeva E.M., Kolesnik A.L., A.Yu. Stenyashin. Object–component design of software systems. Theoretical and applied questions. Bulletin of Taras Shevchenko National University of Kyiv, Series Physics &Mathematics, Kiev, 2013, special issue, pp. 103–117 (in Ukrainian).
[22] Ekaterina M. Lavrischeva. Assemblling Paradigms of Programming in Software Engineering. Journal of Software Engineering and Applications, vol. 9, no. 6, 2016, pp. 296–317, DOI: 10.4236/jsea.2016.96021.
[23] Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle. In Proc. of the 2015 Science and Information Conference (SAI), Jule 28–30, London, pp. 965–972.
[24] John Hebeler, Matthew Fisher, Ryan Blace, Andrew Perez-Lopez. Semantic Web programming. Willey, 2009, 652 p.
[25] Semantic Web. Representation of data on the World Wide Web based on the RDF standards. URL: http://www.w3.org/2001/sw/.
[26] John D. McGregor, David A. Sykes. Practical Guide to testing of Object–oriented software. Addison–Wesley Professional, 2001, 416 p.
[27] Sayyad A. S., Ingram J., Menzies T., Ammar H. Scalable product line configuration: a straw to break the camel's back. In Proc. of the IEEE/ACM 28th International Conference on Automated Software Engineering (ASE 2013), 2013, pp. 465–474.
[28] Lavrischeva E.M., Ryzhov A.G. The approach to the creation of systems and sites of ready–made resources. In Proc. of the XX All–Russian scientific conference "Scientific service on the Internet", Novorossiysk, 17–22 September 2018, CEUR Workshop Proceedings, vol. 2260. pp. 321–345 (in Russian).
[29] Kozin S.V. Configuration build of the Linux kernel variant for application systems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue3, pp. 161-170 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-9.
[30] E.M. Lavrischeva, A.K. Petrenko. Informatics: Formation of computer software and technologies of software systems. ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 7-30. DOI: 10.15514/ISPRAS-2018-30(5)-1.
[31] Lavrishcheva E. M. Informatics and Computer-70. Analysis and development aspects. Conference paper, Ivannikov ISP RAS Open Conference, 2018.
[32] E. M. Lavrischeva. The Scientific Basis of Software Engineering. International Journal of Applied and Natural Sciences (IJANS), vol. 7, issue 5, 2018, pp. 15–32.
[33] 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. ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 99-120 (in Russian). DOI: 10.15514/ISPRAS-2018-30(3)-8.