ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана, ОАО "Концерн “Сириус"

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/pi.html E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

№ 6
June
2015

Published since September 2010

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/pi.html, e-mail:prin@novtex.ru

УДК 004.451

А. Н. Годунов, канд. физ.-мат. наук, зав. отделом, e-mail: nkag@niisi.ras.ru,
В. А. Солдатов, канд. техн. наук, ст. науч. сотр., e-mail: nkvalera@niisi.ras.ru,
Научно-исследовательский институт системных исследований РАН (НИИСИ РАН), г. Москва

Спецификация ARINC 653 и ее реализация в операционной системе реального времени Багет 3

Описаны основные положения спецификации ARINC 653, отмечены ее особенности и отличия от стандарта POSIX. Проанализированы вопросы соответствия положений ARINC 653 тем положениям, которые реализованы в операционной системе реального времени Багет 3.

Ключевые слова: *реальное время, операционная система, надежность, POSIX, ARINC 653*

Спецификация ARINC 653 определяет интерфейс (APEX — *APplication EXecutive*) прикладного программного обеспечения (ПО) с операционной системой (ОС) на основе концепции, обеспечивающей временное и пространственное разделение ресурсов. Эта спецификация разрабатывалась для использования в авионике, но может применяться в системах управления другого назначения с повышенными требованиями к обеспечению безопасности.

Первая версия ARINC 653 вышла в октябре 1996 г. В дальнейшем было опубликовано несколько дополнений, последние из которых датируются декабрем 2014 г. Современное описание спецификации содержится в документах, состоящих из введения и пяти частей [1—6].

Во введении [1] дан общий обзор всех документов, которые относятся к спецификации ARINC 653, представлены сведения о выпущенных и планируемых редакциях документов, указано назначение каждой из частей спецификации. В первой части [2] описаны стандартный (базовый) интерфейс для программных систем в авионике и определены структуры XML-таблиц конфигурирования для прикладных систем, разработанных на основе этого интерфейса. Во второй части [3] описаны дополнительные возможности (расширения) к базовому интерфейсу и структуры XML-таблиц конфигурирования, соответствующие системам с расширенным интерфейсом. В третьей части [4] определены состав и порядок выполнения тестовых процедур для проверки соответствия интерфейса спецификации ARINC 653. В четвертой части [5] определено подмножество интерфейсов, описанных в первой части спецификации. И наконец, в пятой части [6] определен интерфейс для драйверов и пакета поддержки модуля. В данной работе будут рассмотрены основные части спецификации и некоторые особенности ее реализации в операционной системе реального времени (ОСРВ) Багет 3 [7, 8], которая была разработана в НИИСИ РАН.

При сравнении спецификации ARINC 653 со стандартом POSIX (*Portable Operating System Interface*) [9], который также описывает интерфейс между прикладными программами и ОС, можно отметить следующие основные отличия.

- POSIX создавался в первую очередь для ОС общего назначения, а ARINC 653 изначально был ориентирован на ОСРВ с высокими требованиями к надежности их эксплуатации.
- Число интерфейсных функций в POSIX составляет свыше тысячи, а в ARINC 653 — на порядок меньше.
- Терминология, используемая в ARINC 653, отличается от терминологии, принятой в стандарте POSIX. В ARINC 653 единицей планирования ресурсов является раздел (*partition*). В стандарте POSIX этому термину соответствует понятие "процесс" (*process*). В рамках одного раздела возможно параллельное выполнение нескольких процессов. Тогда как в POSIX в рамках одного процесса возможно параллельное выполнение нескольких потоков управления (*thread*). В настоящей статье будем использовать терминологию, принятую в спецификации ARINC 653.
- В спецификации ARINC 653 каждый раздел может находиться в одном из предопределенных состояний (режимов), аналогичное понятие в POSIX отсутствует.
- Еще одной особенностью спецификации ARINC 653 являются жестко определенные требования к порядку выделения ресурсов, требуемых разделу, все необходимые ресурсы выделяются только на этапе инициализации раздела. Однажды выделенные ресурсы не освобождаются, пока работает раздел. Отсутствуют средства для динамического выделения оперативной памяти, вся память, необходимая для выполнения программ раздела, определяется при конфигурировании. При программировании в стандарте POSIX обычной практикой является динамическая работа с ресурсами, когда они многократно запрашиваются и освобождаются, что может приводить к ряду неприятных последствий, например к фрагментации памяти.
- В ARINC 653 для всех объектов ОС определено имя объекта, идентификатор объекта и однотипный интерфейс функций, обслуживающих объект, для

каждого объекта существует функция создания, опроса идентификатора по имени, опроса состояния, плюс две-три функции, специфичные для каждого объекта. В соответствии со стандартом POSIX объекты ОС могут быть как именованными, так и без имени, а интерфейсы функций работы с различными объектами часто непохожи.

- В ARINC 653 детально прописан порядок обработки ошибок. Ошибки, в зависимости от их уровня и этапа выполнения раздела, могут обрабатываться либо средствами ОС, либо прикладной программой. При конфигурировании каждому типу ошибки присваивается соответствующий ей уровень и определяется реакция системы на ошибку каждого уровня в зависимости от этапа выполнения раздела. Прикладная программа может обрабатывать только ошибки уровня процесса. Для этой цели используется "обработчик ошибок" (*error handler*) — специальный процесс раздела с наивысшим приоритетом, который создается при инициализации раздела вызовом соответствующей функции. Аналогичных понятий в POSIX нет.
- В ARINC 653 все требуемые ресурсы и взаимосвязи между отдельными частями системы определяются системным интегратором при конфигурировании системы. Структура таблиц конфигуратора описывается в спецификации ARINC 653 на языке XML-схема. В POSIX понятие конфигурирования системы отсутствует.

Порождение и состояния разделов

Спецификация ARINC 653 предусматривает как пользовательские, так и системные разделы. Интерфейс пользовательских разделов должен соответствовать требованиям спецификации, а на интерфейс системных разделов не накладывается никаких ограничений. При реализации ARINC 653 в ОСРВ Багет 3 для системных разделов используется интерфейс стандарта POSIX в той мере, в какой это допускается спецификацией ARINC 653.

По спецификации ARINC 653 разделы и выделяемые им ресурсы определяются системным интегратором при конфигурировании системы. Загрузка и создание разделов возможны только на этапе инициализации модуля. После завершения этапа инициализации порождение разделов невозможно, но возможен рестарт (останов и повторный запуск) пользовательских разделов. В ОСРВ Багет 3 рестарт системных разделов возможен только вместе с рестартом модуля.

Каждый пользовательский раздел может находиться в одном из перечисленных далее четырех предопределенных режимов (состояний).

- COLD_START — холодный старт. Работа пользовательского раздела начинается с этапа инициализации раздела, когда выполняется только один процесс. На этом этапе запрещено планирование, выполняется только главная функция раздела. Эта функция создает процессы и другие используемые разделом объекты ОС, такие как семафоры, доски объявлений, очереди сообщений, описатели событий, порты. Создание всех этих объектов на других этапах невозможно.
- WARM_START — горячий старт. Этот режим совпадает с предыдущим режимом, но могут быть отличия в начальном состоянии и контексте, зависящие от аппаратуры, например, в этом режиме нет необходимости копирования кода из энергонезависимой памяти в ОЗУ. Существование отличий между режимами холодного и горячего старта для приложений определяется разработчиками ОС.
- NORMAL — рабочий режим. Этап инициализации раздела полностью завершен, и планировщик процессов раздела активен. Все созданные процессы выполняются или ожидают требуемых для своего выполнения ресурсов.
- IDLE — режим простоя, когда не выполняется ни один процесс раздела.

В рабочий режим раздел переводится прикладной программой в случае успешного завершения этапа инициализации. В противном случае раздел переводится в режим простоя или в режим повторной инициализации (COLD_START или WARM_START). Для смены состояния раздела предназначена функция SET_PARTITION_MODE(). Отметим, что реализация этой части спецификации в ОСРВ Багет 3 соответствует стандарту.

Планирование

Планирование выполнения разделов в спецификации ARINC 653 периодическое и определяется расписанием. При составлении расписания определяется длительность основного периода (*major frame*), который разбивается на непересекающиеся временные интервалы (окна). После завершения основного периода он повторяется снова до окончания работы системы или до смены расписания. Для каждого окна назначается раздел, который может выполняться в этом временном интервале. На рис. 1 приведен условный пример расписания.

В описании базового интерфейса спецификации [2] отмечено, что для каждого модуля может быть создано только одно расписание, в расширении спецификации [3] указано, что на разных стадиях работы модуля

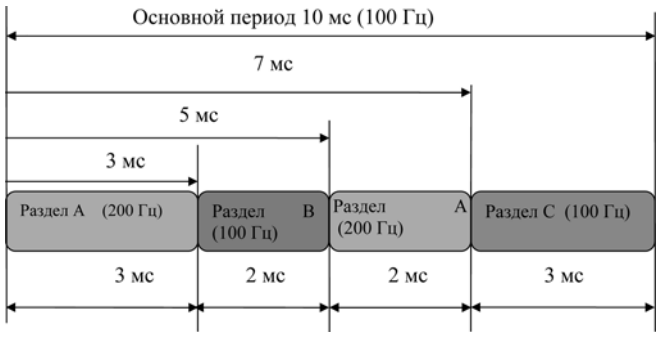


Рис. 1. Пример расписания с четырьмя окнами и тремя разделами

могут использоваться разные расписания. У каждого расписания свой основной период, свой список окон, а у каждого окна свой раздел, который может выполняться в пределах этого окна. Каждому расписанию присваивается имя, отличное от имен других расписаний.

Все расписания определяются системным интегратором при конфигурировании системы. При конфигурировании также задается расписание, которое будет использоваться при старте системы.

Для управления расписаниями окон предназначены следующие функции:

SET_MODULE_SCHEDULE() — установка расписания;

GET_MODULE_SCHEDULE_STATUS() — опрос расписания;

GET_MODULE_SCHEDULE_ID() — опрос идентификатора расписания по имени.

Функция SET_MODULE_SCHEDULE() выдает запрос на смену расписания и немедленно возвращает управление. Фактическая смена расписания проводится в конце текущего основного периода (*major frame*).

При смене расписания окон разделы, раньше не выполнявшиеся (так как не были представлены в использовавшихся расписаниях), но которые должны выполняться в устанавливаемом расписании, инициализируются. Те разделы, которые в момент смены окон находились в режиме NORMAL и представлены в новом расписании, будут перезапущены в соответствии с атрибутом конфигурации ScheduleChangeAction, который может принимать следующие значения:

- COLD_START — холодный рестарт;
- WARM_START — горячий рестарт;
- IGNORE — не проводить рестарт.

Функция GET_MODULE_SCHEDULE_STATUS() позволяет получить следующую информацию о планировании разделов:

- время последней смены расписания;
- идентификатор используемого сейчас расписания;
- идентификатор следующего расписания (указанного функцией SET_MODULE_SCHEDULE()).

Реализация расписания в ОСРВ Багет 3 в основном совпадает с расширением спецификации ARINC 653. Однако имеются отличия, а именно: отдельному окну может быть приписано более одного раздела. В этом случае, если ни один процесс первого в списке раздела не является готовым к выполнению, управление передается готовому к выполнению процессу из наиболее приоритетного раздела. Приоритет разделов при этом определяется порядковым номером в списке.

Управление выполнением процессов и служба времени

Для организации (псевдо) параллельных вычислений в спецификации ARINC 653 используются процессы (*process*). Все процессы одного раздела используют общее адресное пространство. Каждый процесс должен быть создан на этапе инициализации раздела. На этом этапе планирование запрещено и может выполняться только функция инициализации раздела. Согласно спецификации процесс может находиться в одном из следующих состояний:

- DORMANT — неактивный процесс (спящий);
- WAITING — ожидающий процесс;
- READY — работоспособный процесс;
- RUNNING — выполняемый (текущий) процесс.

Создание процесса осуществляется функцией CREATE_PROCESS(), при этом определяются следующие атрибуты создаваемого процесса:

- имя процесса;
- адрес точки входа процесса;
- размер стека процесса;
- базовый (начальный) приоритет процесса;
- период процесса;
- лимит времени выполнения процесса;
- действия в случае превышения лимита времени.

При создании процесса он получает уникальный идентификатор, который используется для указания этого процесса.

После создания процесс находится в неактивном состоянии (DORMANT) до тех пор, пока он не будет запущен функцией START() или DELAYED_START(). Если одна из этих функций была вызвана на этапе инициализации раздела, то процесс переходит в состояние ожидания (WAITING) перехода раздела в рабочий режим (NORMAL). Если процесс был запущен, когда раздел находился в рабочем режиме, то процесс становится работоспособным (READY) немедленно или через определенное время.

Уничтожить процесс невозможно, но можно его перевести в неактивное состояние (DORMANT) функциями STOP() и STOP_SELF(). В состояние DORMANT процесс также переходит при возврате управления из главной функции процесса (ее адрес указывается среди атрибутов процесса при его создании). Неактивный (DORMANT) процесс не может выполняться до тех пор, пока он не будет снова запущен функцией START() или DELAYED_START(). После запуска (старта) процесс начинает выполняться с самого начала (с точки входа).

В состоянии WAITING процесс может находиться по двум причинам. Первая причина — одна из следующих: ожидание перехода раздела в рабочий режим; ожидание истечения интервала времени; ожидание начала следующего периода (для периодических процессов); ожидание семафора; ожидание события; ожидание сообщения. Вторая причина — приостановка процесса функцией SUSPEND() или SUSPEND_SELF(). Возможна комбинация этих двух причин, например, процесс, ждущий семафора, может быть приостановлен функцией SUSPEND().

Приостановленный процесс не может выполняться, пока его работа не будет возобновлена функцией RESUME(). Если приостановить работоспособный процесс, то он исключается из очереди работоспособных процессов. Он снова станет работоспособным, если к нему применить функцию RESUME().

Если процесс в момент приостановки находился в ожидании события, ресурса или интервала времени, то он продолжает ожидание. Требуемый ресурс может быть выделен процессу, несмотря на то что он приостановлен.

Работоспособные процессы (READY), а также выполняемый процесс (RUNNING) находятся в очереди, упорядоченной по приоритетам. В начале очереди находятся более приоритетные процессы, в конце — менее приоритетные. Процессы, имеющие равный приоритет, упорядочены по времени установки в очередь (ранее установленные в очередь процессы находятся ближе к началу очереди).

Получить состояние "выполняемый процесс" могут только работоспособные процессы. При планировании для исполнения выбирается процесс, находящийся в начале очереди работоспособных процессов (наиболее приоритетный процесс, а среди процессов с равным приоритетом тот, который раньше был установлен в очередь). Если планирование не запрещено, то оно проводится всякий раз, когда выполняемый процесс перестает быть первым в очереди работоспособных процессов.

Начальный (базовый) приоритет процесса указывается при его порождении. Изменить приоритет процесса можно функцией SET_PRIORITY(). Если приоритет устанавливается у выполняемого или работоспособного процесса, то он извлекается из очереди работоспособных процессов и вновь в нее устанавливается в соответствии с новым значением приоритета. Если новый приоритет процесса равен текущему, то процесс станет последним среди процессов с тем же приоритетом. Если в результате выполняемый процесс перестанет быть первым в очереди работоспособных, то будет проведено перепланирование процессов (если оно не запрещено).

Если планирование запрещено функцией LOCK_PREEMPTION(), то выполняемый процесс будет выполняться до тех пор, пока планирование не будет разрешено функцией UNLOCK_PREEMPTION(), даже если существуют работоспособные процессы с большим приоритетом. Запрещение планирования влияет на работу функций, которые могут перевести выполняемый процесс в неработоспособное состояние (WAITING). В таких случаях эти функции не переводят процессы в состояние WAITING, а немедленно возвращают управление с соответствующим кодом ошибки.

В соответствии с ARINC 653 существуют два вида процессов: периодические и непериодические. Как периодические, так и непериодические процессы создаются функцией CREATE_PROCESS(). При создании периодического процесса следует указать период процесса и лимит времени. Для непериодического процесса период должен иметь значение INFINITE_TIME_VALUE.

Лимит времени ограничивает время работы процесса. Для периодического процесса время работы отсчитывается с начала периода. Лимит времени не может быть больше периода. Изменить лимит времени можно с помощью функции REPLENISH(). Для организации периодической работы процесса используется функция PERIODIC_WAIT(). Она приостанавливает процесс до начала следующего периода.

Типичный периодический процесс работает по следующей схеме. Процесс выполняет действия, которые нужно сделать в данном периоде, и вызывает функцию PERIODIC_WAIT() для ожидания начала следующего периода. Далее выполняются действия следующего периода, вызывается функция PERIODIC_WAIT() и т. д. Схематично текст главной функции такого периодического потока имеет следующий вид:

```
void main_cycle() {
RETURN_CODE_TYPE ErrorCode; // Код возврата
while(1) {
    ((ErrorCode = period_func(...)) != NO_ERROR) break ;
    PERIODIC_WAIT(&ErrorCode) ;
    if (ErrorCode != NO_ERROR) break;
    }
    ...
}
```

Наличие в спецификации периодических процессов существенно облегчает прикладное программирование систем управления для случая, когда требуется циклическая обработка внешних воздействий за ограниченный промежуток времени.

Далее перечислены функции, предназначенные для управления выполнением процессов (последние четыре функции относятся к функциям службы времени).

CREATE_PROCESS	Создание процесса
GET_PROCESS_ID	Опрос идентификатора процесса по имени
GET_PROCESS_STATUS	Опрос статуса процесса
SET_PRIORITY	Установка приоритета процесса
SUSPEND_SELF	Приостановка выполняемого процесса
SUSPEND	Приостановка процесса
RESUME	Возобновление выполнения приостановленного процесса
STOP_SELF	Останов выполняемого процесса
STOP	Останов выполнения процесса
START	Запуск процесса на выполнение
DELAYED_START	Отложенный запуск процесса на выполнение
LOCK_PREEMPTION	Запрещение переключения процессов
UNLOCK_PREEMPTION	Разрешение переключения процессов
GET_MY_ID	Опрос идентификатора выполняемого процесса
TIMED_WAIT	Временная приостановка процесса

PERIODIC_WAIT . . . . . Ожидание начала следующего периода
GET_TIME . . . . . Опрос текущего системного времени
REPLENISH . . . . . Изменение лимита времени выполнения процесса

Реализация этой части спецификации в ОСРВ Багет 3 соответствует стандарту.

Взаимодействие между разделами

Согласно спецификации ARINC 653 пользовательские разделы могут взаимодействовать между собой только путем передачи сообщений через каналы. Такое ограничение делает разделы слабо взаимосвязанными. Это в значительной мере уменьшает влияние ошибок в одних разделах на работу других разделов, а также упрощает восстановление работоспособности системы в случае обнаружении ошибок при работе пользовательских разделов. Если в результате обработки ошибки, возникшей при выполнении одного из пользовательских разделов, этот раздел будет повторно запущен (проведен рестарт), то это может привести лишь к потере нескольких сообщений другими разделами.

Доступ к каналам осуществляется через порты. У каждого канала имеется только один порт, передающий сообщения, а также один или несколько портов, принимающих сообщения. Данные в канале могут передаваться только в одном направлении.

Каналы могут использоваться для передачи сообщений: между разделами, выполняемыми на одном и том же процессорном модуле; между разделами, выполняемыми на разных процессорных модулях; между разделами и внешними устройствами. В первом случае передача данных проводится средствами ОС, во втором и третьем случаях — драйверами и ОС.

Каналы могут работать в двух режимах: с очередью сообщений (QUEUING) и без очереди сообщений (SAMPLING). Интерфейс взаимодействия прикладных программ с каналами не зависит от способа передачи данных по каналам, но зависит от режима работы канала. Для каждого режима работы используется свой набор функций.

Независимость интерфейса от способа передачи данных значительно повышает мобильность прикладных программ и позволяет системному интегратору выбирать способы передачи данных на этапе конфигурирования системы без внесения изменений в прикладные программы.

Каналы и порты должны быть описаны при конфигурировании системы. Указанные при конфигурировании параметры каналов (используемые порты, режим работы, направление передачи данных и др.) нельзя изменить во время работы системы.

Канал, работающий в режиме с очередью сообщений, имеет один порт приема сообщений и один порт отправки сообщений. Порт отправки сообщений содержит очередь отправляемых сообщений, а порт приема сообщений — очередь принимаемых сообщений. Размер очередей ограничен и указывается при создании порта. Каждое отправляемое сообщение доставляется пользователю и может быть получено только один раз.

Канал, работающий в режиме без очереди сообщений, может иметь только один порт отправки сообщений и один или несколько портов приема сообщений. Каждое сообщение, получаемое портом приема, замещает ранее полученные сообщения (они просто теряются). Таким образом, из порта всегда считывается сообщение, полученное последним. Прочитанное сообщение считается годным, если с момента поступления его в порт приема до момента чтения из порта прошло времени не больше, чем период обновления порта.

Если при отправке сообщения оказалось, что по каким-то причинам предыдущее сообщение не было отправлено, то новое сообщение замещает старое. Таким образом, отправка сообщения в порт всегда возможна и прикладная программа всегда может выводить сообщения в порт с нужной частотой (это не относится к процессу передачи данных от порта приема к порту передачи).

Далее перечислены функции, обеспечивающие интерфейс прикладной программы с портами.

CREATE_SAMPLING_PORT . . . . . Создание порта без очереди сообщений
WRITE_SAMPLING_MESSAGE . . . . Вывод сообщения в порт без очереди сообщений
READ_SAMPLING_MESSAGE . . . . . Ввод сообщения из порта без очереди сообщений
GET_SAMPLING_PORT_ID. . . . . Опрос идентификатора порта без очереди сообщений
GET_SAMPLING_PORT_STATUS . . . Опрос состояния порта без очереди сообщений
CREATE_QUEUING_PORT . . . . . Создание порта с очередью сообщений;
SEND_QUEUING_MESSAGE . . . . . Отправка сообщения в порт с очередью сообщений
RECEIVE_QUEUING_MESSAGE . . . Получение сообщения из порта с очередью сообщений
GET_QUEUING_PORT_ID. . . . . Опрос идентификатора порта с очередью сообщений
GET_QUEUING_PORT_STATUS . . . . Опрос состояния порта с очередью сообщений
CLEAR_QUEUING_PORT . . . . . Сброс порта с очередью сообщений

В расширении спецификации [3] содержится также описание SAP (*Service Access Point*) портов и списков портов.

Порты SAP являются разновидностью портов с очередью сообщений. Однако они отличаются тем, что в функции отправки сообщения может быть ука-

зан адрес получателя, а в функции получения сообщения получен адрес отправителя. Для основных функций отправки и получения сообщений формат адреса соответствует формату адреса сокета в POSIX, а для дополнительных функций он может быть определен по усмотрению разработчиков ОС.

Списки портов позволяют получать сообщения, прибывающие по нескольким каналам в контексте одного процесса, без постоянного опроса портов. Списки портов могут содержать только порты с очередью сообщений, предназначенные для получения сообщений. Создание списка портов, а также добавление портов в список возможно только на стадии инициализации раздела. Изменение уровня активности порта в списке возможно как на стадии инициализации, так и в нормальном режиме работы.

Далее представлены функции, предназначенные для работы с этими объектами.

CREATE_SAP_PORT	Создание SAP-порта
SEND_SAP_MESSAGE	Отправка сообщения в SAP-порт
RECEIVE_SAP_MESSAGE	Получение сообщения из SAP-порта
GET_SAP_PORT_ID	Опрос идентификатора SAP-порта
GET_SAP_PORT_STATUS	Опрос состояния SAP-порта
SEND_EXTENDED_SAP_MESSAGE	Отправка сообщения в SAP-порт с другим форматом адреса
RECEIVE_EXTENDED_SAP_MESSAGE	Получение сообщения из SAP-порта с другим форматом адреса
CREATE_QUEUING_PORT_LIST	Создание списка портов
ADD_PORT_TO_QUEUING_PORT_LIST	Внесение порта в список портов
SET_PORT_ACTION_IN_QUEUING_PORT_LIST	Установка уровня активности порта в списке портов
GET_QUEUING_PORT_LIST_STATUS	Опрос состояния порта в списке
RECEIVE_MESSAGE_FROM_QUEUING_PORT_LIST	Получение сообщения по списку портов

Реализация взаимодействия между разделами в ОСРВ Багет 3 соответствует спецификации ARINC 653. Однако следует отметить, что не реализованы функции отправки и получения сообщений для SAP-портов с другим форматом адреса.

Синхронизация

В спецификации ARINC 653 для синхронизации между различными процессами, которые выполняются в одном разделе, предназначены семафоры (SEMAPHORE) и события (EVENT).

Семафоры спецификации ARINC 653 во многом схожи с семафорами POSIX, но есть и существенные отличия:

- для работы с семафорами спецификациями ARINC и POSIX определяются различные интерфейсы, например, функция WAIT_SEMAPHORE в зависимости от значения параметров покрывает возможности функций sem_wait(), sem_timedwait() и sem_trywait() стандарта POSIX;
- семафоры ARINC, созданные в рамках одного раздела, могут использоваться только процессами данного раздела и не влияют на выполнение процессов других разделов;
- в ARINC дисциплина обслуживания очереди к семафору определяется пользователем при создании семафора (приоритетная или FIFO);
- интерфейс ARINC позволяет реализовать как целочисленные, так и двоичные (бинарные) семафоры, тогда как в стандарте POSIX бинарные семафоры не предусмотрены;
- каждый семафор в ARINC-спецификации имеет имя, уникальное в пределах данного раздела;
- создавать семафор в ARINC-спецификации можно только на этапе инициализации, и созданный семафор нельзя уничтожить (удалить).

При создании семафора указывают его основные параметры: имя семафора; начальное значение счетчика семафора; максимальное значение счетчика семафора; дисциплина обслуживания очереди к семафору (приоритетная или FIFO).

Каждому семафору назначается уникальный идентификатор, который используется другими функциями для указания семафора. Основными операциями при работе с семафорами являются захват и освобождение семафора. В обеих операциях используется счетчик семафора — целое число, начальное значение которого определяется при создании семафора.

Функции, предназначенные для работы с семафорами и событиями, представлены далее.

CREATE_SEMAPHORE	Создание семафора
WAIT_SEMAPHORE	Захват семафора
SIGNAL_SEMAPHORE	Освобождение семафора
GET_SEMAPHORE_ID	Опрос идентификатора семафора по имени
GET_SEMAPHORE_STATUS	Опрос состояния семафора
CREATE_EVENT	Создание описателя события
SET_EVENT	Уведомление о событии
RESET_EVENT	Сброс описателя события
WAIT_EVENT	Ожидание события
GET_EVENT_ID	Опрос идентификатора описателя события
GET_EVENT_STATUS	Опрос состояния описателя событий

При захвате семафора проверяется счетчик семафора. Если значение счетчика больше нуля, то оно

уменьшается на 1, семафор считается захваченным и процесс продолжает работу. Если счетчик равен 0, то это означает, что в настоящее время семафор не может быть захвачен. В таком случае функция WAIT_SEMAPHORE() либо приостанавливает процесс и устанавливает его в очередь к семафору, либо возвращает управление с соответствующим кодом возврата.

Функция WAIT_SEMAPHORE() позволяет ограничить время нахождения потока в очереди. По исчерпании указанного лимита времени процесс становится работоспособным, даже если семафор захватить не удалось. При этом функция WAIT_SEMAPHORE() возвращает соответствующий код. Если указан нулевой лимит времени и семафор захватить не удалось, то функция немедленно возвращает управление с соответствующим кодом возврата.

В очереди к семафору может находиться одновременно несколько процессов. Если используется приоритетная очередь, то процесс устанавливается в очередь после всех процессов с большим или равным приоритетом, но перед всеми процессами с меньшим приоритетом. Таким образом, процессы, находящиеся в очереди, упорядочены по приоритетам, а процессы, имеющие равный приоритет, упорядочены по времени установки в очередь.

Если для очереди используется дисциплина обслуживания FIFO, то процесс устанавливается всегда в конец очереди. Таким образом, все процессы, находящиеся в FIFO-очереди, упорядочены по времени установки в очередь.

При освобождении семафора функция SIGNAL_SEMAPHORE() проверяет, пуста ли очередь к семафору. Если очередь не пуста, то из нее извлекается первый поток и объявляется работоспособным (если он не был ранее приостановлен функцией SUSPEND()). При этом значение счетчика семафора не меняется. Если же очередь к семафору пуста и значение счетчика семафора меньше максимального (указанного при создании семафора), то значение счетчика увеличивается на 1. Если счетчик уже достиг максимального значения, то его значение не изменяется.

Последнее свойство позволяет реализовать двоичные (бинарные) семафоры. Для этого при создании семафора достаточно определить максимальное значение счетчика семафора равным 1. Напомним, что двоичный семафор может принимать только два значения — 0 (семафор занят) и 1 (семафор свободен). Двоичный семафор можно захватить только тогда, когда он свободен. Если освобождаемый двоичный семафор уже свободен, то он не меняет своего состояния.

Описываемый в спецификации ARINC 653 аппарат событий (EVENT) предназначен для оповещения (одного или нескольких) процессов о событии. Описатель события может находиться в одном из двух состояний:

- сброшен (событие не произошло, DOWN);
- взведен (событие произошло, UP).

Для обрабатываемого события или серии однотипных событий прикладная программа создает описатель события во время инициализации раздела с помощью функции CREATE_EVENT(). При создании описателя события ему назначается уникальный идентификатор, который используется другими функциями для указания описателя события. После создания описателя события он находится в состоянии "сброшен" (событие не произошло).

Функция WAIT_EVENT() проверяет, произошло ли указанное событие. Если событие произошло (описатель находится в состоянии "взведен"), то процесс продолжает работу.

Функция SET_EVENT() сообщает о том, что событие произошло. При этом описатель события переходит в состояние "взведен", а все процессы, ожидающие данного события, переходят в состояние "работоспособен", за исключением приостановленных. Если описатель события уже находился в состоянии "взведен", то никаких действий не проводится.

Функция RESET_EVENT() переводит описатель события в состояние "сброшен" (DOWN). Она применяется в тех случаях, когда один описатель события используется для серии однотипных событий. Ее вызывают при переходе от предыдущего события к следующему.

Отметим, что реализация этой части спецификации в ОСРВ Багет 3 соответствует стандарту.

Очереди сообщений и доски объявлений

В спецификации ARINC 653 для передачи данных между различными процессами одного раздела предназначены очереди сообщений (BUFFER) и доски объявлений (BLACKBOARD).

Основное отличие очередей сообщений от досок объявлений состоит в том, что сообщения, отправленные в очередь, не теряются. Они находятся в очереди до тех пор, пока не будут прочитаны. Каждое сообщение может быть прочитано только один раз.

Доска объявлений может содержать не более одного сообщения. При выводе сообщения новое сообщение замещает уже имеющееся. После чтения сообщение не удаляется и может быть прочитано несколько раз. Как очереди сообщений, так и доски объявлений могут использоваться несколькими процессами (одного раздела) как на ввод, так и на вывод.

Очереди сообщений ARINC имеют много общего с очередями сообщений POSIX, но есть и следующие отличия:

- ARINC и POSIX предлагают различный интерфейс для очередей сообщений;
- каждая очередь сообщений ARINC может использоваться только процессами одного раздела (что позволяет реализовать очереди сообщений ARINC более эффективно);
- дисциплина обслуживания очереди процессов к очереди сообщений ARINC определяется пользователем (приоритетная или FIFO);
- ARINC не поддерживает приоритеты сообщений;
- создавать очередь сообщений в соответствии с ARINC можно только на этапе инициализации, и созданную очередь нельзя уничтожить (удалить).

При создании очереди сообщений указывают ее основные параметры: имя очереди сообщений; максимальный размер сообщения; максимальное число сообщений в очереди; дисциплину обслуживания очередей потоков (приоритетная или FIFO).

Каждой очереди сообщения при ее создании назначается уникальный идентификатор, который используется другими функциями для указания очереди сообщений. Очереди сообщений предназначены для обмена данными между процессами. Основными операциями являются отправка и получение сообщения.

При отправке сообщения функция SEND_BUFFER() проверяет, пуста ли очередь процессов, ждущих сообщений. Если очередь не пуста, то выбирается первый процесс из этой очереди. Выводимое сообщение передается этому процессу, а сам процесс объявляется работоспособным. Такой подход позволяет сократить время на пересылках сообщений. Если же очередь процессов, ждущих сообщений, пуста и в очереди сообщений есть свободное место, то выводимое сообщение помещается в конец очереди сообщений. Если очередь сообщений переполнена, то функция SEND_BUFFER() либо приостанавливает процесс и устанавливает его в очередь процессов, ждущих свободного места, либо возвращает управление с соответствующим кодом возврата.

Функция SEND_BUFFER() позволяет ограничить время нахождения процесса в очереди. По исчерпании указанного лимита времени процесс становится работоспособным, даже если сообщение не удалось поместить в очередь. При этом функция SEND_BUFFER() возвращает соответствующий код. Если указан нулевой лимит времени, а очередь сообщений переполнена, то функция немедленно возвращает управление с соответствующим кодом возврата.

Для получения сообщений из очереди используется функция RECEIVE_BUFFER(). Вначале она проверяет, есть ли сообщения в очереди. Если очередь не пуста, то выбирается первое сообщение в очереди и процесс продолжает работу. Если же очередь сообщений пуста, то функция RECEIVE_BUFFER() либо приостанавливает процесс и устанавливает его в очередь процессов, ждущих сообщения, либо возвращает управление с соответствующим кодом возврата. Место процесса в очереди обусловливается дисциплиной обслуживания очереди, определяемой при создании очереди сообщений.

При создании доски объявлений указывают ее основные параметры: имя доски объявлений и максимальный размер сообщения.

При создании доски объявлений ей назначается уникальный идентификатор. Создание доски объявлений возможно только на этапе инициализации раздела. После создания доска объявлений пуста (не содержит сообщения).

Функция DISPLAY_BLACKBOARD() позволяет вывести сообщение на доску объявлений. Если доска уже содержала сообщение, то выводимое сообщение замещает уже имеющееся. Если доска не содержала сообщения, то сообщение помещается на доску объявлений, а процессы, ждущие сообщения, объявляются работоспособными. Эта функция никогда не переводит процесс в неработоспособное состояние.

Функция READ_BLACKBOARD() предназначена для чтения сообщения. Вначале функция проверяет, есть ли сообщение на доске объявлений. Если сообщение имеется, то функция считывает это сообщение (доска объявлений всегда содержит не более одного сообщения) и процесс продолжает работу. В противном случае функция либо приостанавливает процесс, либо возвращает управление с соответствующим кодом возврата. Если процесс перейдет в работоспособное состояние, так как было выведено сообщение, то при его запуске будет считано сообщение, выведенное на доску объявлений последним, даже если оно к этому моменту будет стерто.

Функции, предназначенные для работы с очередями сообщений и досками объявлений, представлены далее.

CREATE_BUFFER	Создание очереди сообщений
SEND_BUFFER	Отправка сообщения в очередь
RECEIVE_BUFFER	Получение сообщения из очереди
GET_BUFFER_ID	Опрос идентификатора очереди сообщений
GET_BUFFER_STATUS	Опрос состояния очереди сообщений
CREATE_BLACKBOARD	Создание доски объявлений
DISPLAY_BLACKBOARD	Вывод сообщения на доску объявлений
READ_BLACKBOARD	Чтение сообщения с доски объявлений
CLEAR_BLACKBOARD	Очистка доски объявлений
GET_BLACKBOARD_ID	Опрос идентификатора доски объявлений
GET_BLACKBOARD_STATUS	Опрос состояния доски объявлений

Отметим, что реализация этой части спецификации в ОСРВ Багет 3 соответствует стандарту.

Книги записей

В расширении спецификации ARINC 653 описываются книги записей (*logbook*), предназначенные для хранения сообщений (записей) между сеансами работы (при выключении питания). Одновременно можно использовать несколько книг записей, но каждая книга записей может использоваться только одним разделом. Используемые книги записей должны быть описаны при конфигурировании системы.

Для использования книги записей она должна быть предварительно инициализирована функцией CREATE_LOGBOOK(). Эта функция может быть вызвана только на стадии инициализации раздела. При

вызове функции указывают основные параметры книги записей: имя книги записей; максимальный размер сообщения (записи); максимальное число сообщений (записей) в книге; размер буфера в оперативной памяти.

При создании книги записей:

- область флэш-памяти связывается с создаваемым объектом и все сообщения, ранее записанные в эту область, становятся доступны для чтения;
- в оперативной памяти создается промежуточный буфер, через который сообщения переносятся во флэш-память;
- создаваемой книге записей назначается уникальный идентификатор.

Получить значение идентификатора книги записей можно также с помощью функции GET_LOGBOOK_ID(), указав ее имя.

Функции записи WRITE_LOGBOOK() и OVERWRITE_LOGBOOK() помещают выводимое сообщение в буфер. Копирование сообщений из буфера во флэш-память проводится не сразу, а по мере возможности. Если буфер заполнен, то функция WRITE_LOGBOOK() теряет сообщение, а функция OVERWRITE_LOGBOOK() записывает выводимое сообщение на место предыдущего сообщения. Про сообщения, помещенные в буфер, но еще не выведенные во флэш-память, говорят, что они находятся в состоянии обработки или в состоянии IN_PROGRESS. Число сообщений, хранящихся в книге записей, ограничено. Это значение указывается при вызове функции CREATE_LOGBOOK(). При попытке записать большее число записей новые записи вытесняют наиболее старые записи, находящиеся в книге. Если при выводе сообщения во флэш-память произошла ошибка (например, было выключено питание и вывод не дошел до конца), то выведенное сообщение будет помечено как отброшенное (ABORTED).

Чтение книги записей проводится с помощью функции READ_LOGBOOK(). Для указания читаемой записи используется индекс. Индекс возрастает от самой свежей записи (выведенной последней) до самой старой записи. Самая свежая запись имеет индекс 0, предыдущая запись — 1 и т. д. Эта функция также позволяет читать выведенные в книгу записей, но еще не записанные во флэш-память (находящиеся в буфере) сообщения. Вместе с записью функция возвращает ее статус:

- COMPLETE — сообщение успешно записано во флэш-память;
- IN_PROGRESS — сообщение находится в буфере;
- ABORTED — были ошибки при записи сообщения во флэш-память.

Далее перечислены функции, предназначенные для работы с книгами записей.

CREATE_LOGBOOK	Создание книги записей
WRITE_LOGBOOK	Вывод сообщения в книгу записей
OVERWRITE_LOGBOOK	Вывод сообщения с замещением в книгу записей
READ_LOGBOOK	Чтение сообщения из книги записей
CLEAR_LOGBOOK	Очистка книги записей
GET_LOGBOOK_ID	Опрос идентификатора книги записей
GET_LOGBOOK_STATUS	Опрос состояния книги записей

Реализация книги записей в ОСРВ Багет 3 соответствует спецификации ARINC 653.

Файловая система

Во второй части спецификации ARINC 653 описан интерфейс файловой системы, который по своим операциям близок к файловой системе стандарта POSIX. Основной целью интерфейса файловой системы ARINC является абстрактный доступ к данным вне зависимости от типа носителя, на котором они располагаются. В качестве запоминающих устройств для хранения данных могут использоваться область оперативной памяти, флэш-память, EPROM (*Erasable Programmable Read Only Memory*), память, доступная по локальной сети, и т. д.

При реализации файловой системы разработчики ОС должны обеспечить средства управления работой с файлами на низком уровне и доступ к данным из различных разделов как на стадии инициализации раздела, так и в нормальном режиме работы раздела. Задержки, возникающие при работе с данными файловой системы, достоверность, целостность и сохранность этих данных также зависят от конкретной реализации.

Каждая функция файловой системы должна включать в себя проверку корректности всех входных параметров, а также контроль за выполнением операций обмена данными с конкретным носителем. Запоминающие устройства для хранения данных должны быть описаны при конфигурировании модуля. Именование файлов близко к именованию файлов в стандарте POSIX, но имеются небольшие отличия:

- полное имя файла содержит в себе наименование тома;
- имена каталогов и файлов нечувствительны к регистру;
- разделителем между именами может быть как символ ‘/’, так и символ ‘\’.

Для каждого тома только один раздел может иметь право на чтение и запись (*read-write*). Сразу несколько разделов могут иметь право на чтение (*read only*) данных с тома. Максимальные размеры томов и права доступа к ним из пользовательских разделов определяются на этапе конфигурирования.

Одной из особенностей интерфейса функций файловой системы является наличие двух переменных для индикации ошибок:

- для кода возврата (RETURN_CODE), эта переменная не очищается;
- для детализации ошибки (ERRNO).

Интерфейс спецификации предписывает, что в случае возникновения ошибочной ситуации каждая функция в переменную RETURN_CODE записывает код ошибки, определяемый спецификацией ARINC, а в переменную ERRNO записывает код ошибки, совпадающий со стандартом POSIX для аналогичной функции.

В таблице представлены функции файловой системы ARINC и соответствующие им функции стандарта POSIX.

Следует отметить, что в ОСРВ Багет 3 данная часть спецификации ARINC пока не реализована.

ARINC-функция	POSIX-функция	Назначение
OPEN_NEW_FILE	create()	Создание файла
OPEN_FILE	open()	Открытие файла
CLOSE_FILE	close()	Закрытие файла
READ_FILE	read()	Чтение файла
WRITE_FILE	write()	Запись файла
SEEK_FILE	lseek()	Позиционирование файла
REMOVE_FILE	unlink()	Удаление файла
RENAME_FILE	rename()	Переименование файла
GET_FILE_STATUS	fstat()	Получение информации о файле
GET_VOLUME_STATUS	statvfs()	Получение информации о файловой системе
RESIZE_FILE	ftruncate()	Изменение размера файла
SYNC_FILE	fsync()	Синхронизация файла
OPEN_DIRECTORY	opendir()	Открытие каталога
CLOSE_DIRECTORY	closedir()	Закрытие каталога
READ_DIRECTORY	readdir()	Чтение каталога
REWIND_DIRECTORY	rewinddir()	Возвращает к началу каталога
MAKE_DIRECTORY	mkdir()	Создание каталога
REMOVE_DIRECTOR	rmdir()	Удаление каталога
SYNC_DIRECTORY	—	Синхронизация каталога
RENAME_DIRECTORY	rename()	Переименование каталога

Обработка ошибок в ARINC 653

Диагностика ошибок проводится аппаратурой, ОС и прикладной программой. Прикладная программа (исполняемая в рамках пользовательского раздела) сообщает ОС об ошибке с помощью функции RAISE_APPLICATION_ERROR(). Реакция на ошибку при выполнении пользовательского раздела зависит от уровня ошибки. Уровень ошибки определяется интегратором при конфигурировании системы в зависимости от типа ошибки и состояния системы. Для пользовательских разделов могут использоваться следующие уровни ошибок:

- ошибки уровня модуля (MODULE);
- ошибки уровня раздела (PARTITION);
- ошибки уровня процесса (PROCESS).

Ошибки уровня модуля обрабатываются ОС. Реакция на эти ошибки определяется системным интегратором при конфигурировании системы. Указываемая реакция может зависеть от идентификатора ошибки и состояния системы, но она не зависит от раздела. Возможны следующие виды реакции на ошибки уровня модуля:

- рестарт модуля (RESET);
- останов модуля (SHUTDOWN);
- игнорирование ошибки (IGNORE).

При возникновении ошибки уровня модуля, для которой в качестве реакции указано RESET, осуществляется перезагрузка данного модуля. Если указана реакция SHUTDOWN, то на данном модуле прекращается выполнение системы до тех пор, пока он не будет заново загружен нажатием клавиши СБРОС/RESET или включением питания. Если указана реакция IGNORE, то ошибка игнорируется.

Ошибки уровня раздела также обрабатываются ОС. Для каждого пользовательского раздела системный интегратор создает свою таблицу ошибок. Возможны следующие виды реакции на ошибки уровня раздела:

- переход раздела в состояние простоя (IDLE);
- горячий рестарт раздела (WARM_START);
- холодный рестарт раздела (COLD_START);
- игнорирование ошибки (IGNORE).

Перевод раздела в режим простоя останавливает работу раздела. Никакие процессы раздела в этом режиме не выполняются, однако память и временные окна, выделенные разделу, сохраняются за данным разделом. Рестарт раздела вызывает останов раздела и запуск его сначала (повторяется инициализация раздела). Если запрошена реакция IGNORE, то ошибка игнорируется.

К ошибкам уровня процесса можно отнести только ошибки, произошедшие при выполнении процесса в рамках пользовательского раздела. Обработка ошибок уровня процесса определяется прикладным программистом с помощью специального процесса обработки ошибок (*error handler*). Создание процесса обработки ошибок с помощью функции CREATE_ERROR_HANDLER возможно только на этапе инициализации раздела.

Процесс обработки ошибок имеет наибольший приоритет. По основной спецификации процесс обработки ошибок активен только в рабочем режиме (NORMAL), а по дополнительной — после вызова функции ENABLE_ERROR_HANDLER(), вне зависимости от режима работы раздела (т. е. возможна обработка ошибок при инициализации раздела).

Старт процесса обработки ошибок проводится ОС при обнаружении ошибки уровня процесса. После запуска процесс обработки ошибок прерывает работу текущего процесса даже тогда, когда переключение процессов запрещено.

Процесс обработки ошибок может остановить, а также остановить и вновь запустить ошибочный процесс с помощью функций STOP() и START(), повторно запустить или остановить раздел с помощью функции SET_PARTITION_MODE(). Работа ошибочного процесса может быть продолжена только в следующих случаях:

- превышение времени работы процесса;
- ошибка обнаружена прикладной программой.

Для вывода сообщений об ошибке процесс обработки ошибок может использовать функцию REPORT_APPLICATION_MESSAGE(), а для получения сведений об ошибке — функцию GET_ERROR_STATUS(). Эта функция, в частности, позволяет получить код ошибки. Допустимые коды ошибок перечислены в спецификации ARINC 653 (стандартизованы). К их числу относятся:

- DEADLINE_MISSED — превышение лимита времени при выполнении периодического процесса;
- APPLICATION_ERROR — ошибка, диагностированная прикладной программой;
- NUMERIC_ERROR — ошибки при выполнении арифметических операций над числами как с фиксированной, так и с плавающей точкой (переполнение, деление на 0 и т. д.);
- ILLEGAL_REQUEST — ошибочное обращение к ОС;
- STACK_OVERFLOW — переполнение стека процесса;
- MEMORY_VIOLATION — нарушение защиты памяти;
- HARDWARE_FAULT — аппаратные сбои (например, ошибка четности);
- POWER_FAIL — прерывание по сбою питания.

Соответствие между идентификаторами ошибок и кодами ошибок устанавливается при конфигурировании системы в системной таблице ошибок. Код ошибки указывается только для состояния системы "выполнение раздела" (в непривилегированном режиме). Идентификаторы ошибок присваиваются ошибкам ОС (не стандартизованы).

Процесс обработки ошибок не может вызывать функции, которые могут перевести его в неработоспособное состояние.

Обработчик ошибок не может быть ни приостановлен, ни остановлен другим процессом. Его приоритет не может быть изменен. По завершении работы процесс обработки ошибок останавливает себя функцией STOP_SELF(). Если процесс обработки ошибок не был создан, то ошибки уровня процесса трактуются как ошибки уровня раздела. В этом случае реакция на ошибку определяется таблицей ошибок раздела. Ошибка при работе процесса обработки ошибок также считается ошибкой уровня раздела.

В ОСРВ Багет 3 реализован основной интерфейс спецификации. Дополнительная функция ENABLE_ERROR_HANDLER() в этой реализации отсутствует. Кроме того, в ОСРВ Багет 3 для сообщения об ошибочной ситуации, кроме функции RAISE_APPLICATION_ERROR(), можно также пользоваться функцией throwException(). Макрооперации tryBegin(), tryCatch(), tryEnd() позволяют облегчить обработку ошибок прикладной программой (аналогично конструкции TRY-CATCH в языке C++). Они могут использоваться для обработки следующих ошибок:

- исключительные ситуации;
- переполнение стека;
- ошибки, обнаруженные прикладной программой или функцией обработки прерываний (если уведомление об ошибке сформировано функцией throwException()).

Исключительная ситуация возникает, если процессор не может корректно выполнить команду (в команде используется недопустимый адрес, проводится деление на 0 и др.). Если ошибка не была обработана с помощью макроопераций tryBegin(), tryCatch(), tryEnd(), то ошибка обрабатывается в соответствии со спецификацией ARINC 653.

Конфигурирование

По спецификации ARINC 653 объекты ОС, которые будут использованы в системе, должны быть предварительно описаны системным интегратором при конфигурировании системы. Реакции на ошибки различных уровней также задаются при конфигурировании.

В спецификации ARINC 653 на языке XML-схема определяются структуры и типы данных, необходимые для конфигурирования любой системы, реализующей эту спецификацию. При описании основных принципов конфигурирования воспользуемся графическим примером из описания спецификации, где корневым элементом является MODULE (Модуль), структура которого представлена на рис. 2.

Атрибут Name элемента MODULE определяет имя модуля. Тип имени (NameType), так же как и типы всех остальных атрибутов, определяется в схеме спецификаций (ARINC 653-Schema type). Кроме того, в корневом элементе содержатся элементы, которые описывают Partitions (разделы), Schedules (расписания) и HealthMonitoring (обработку ошибок).

Элемент Partitions имеет составной тип (PartitionsType), структура которого представлена на рис. 3. Как видно на этом рисунке, для описания разделов используется последовательность (массив) элементов Partition (раздел), каждый из которых содержит четыре

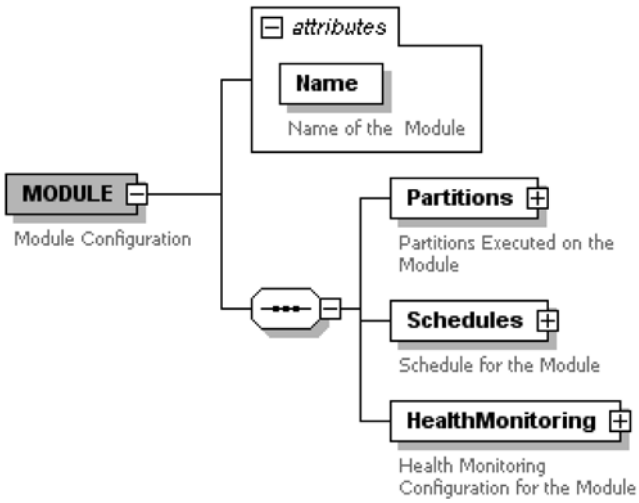


Рис. 2. Корневой элемент XML-схемы ([1, с. 199])

элемента: PartitionDefinition (определение раздела); PartitionPeriodicity (периодичность выполнения раздела); MemoryRegions (области памяти); PartitionPorts (порты раздела).

В описании разделов должен содержаться хотя бы один элемент, описывающий раздел. В описании раздела задаются:

- его имя (Name) и идентификатор (Identifier);
- периодичность выполнения раздела в наносекундах (Period) и его продолжительность (Duration);
- массив областей памяти (MemoryRegions), которые выделяются разделу;
- массив портов (PartionPorts), которые могут быть созданы при выполнении программ раздела.

Для каждой области (MemoryRegion) могут быть заданы: имя области; ее тип (Ram, Flash, Input/output); размер; права доступа и физический адрес памяти. Дополнительно может быть задан атрибут "кэш", который может принимать следующие значения:

- не использовать кэш (Cache_Off);
- не использовать кэш при записи в память (Cache_WriteThrough);
- использовать кэш (Cache_CopyBack).

Такая схема для описания областей памяти для раздела появилась в третьей редакции спецификации. В более ранних редакциях были определены другие типы памяти (Code, Data, Input/Output) и не было дополнительного атрибута "кэш".

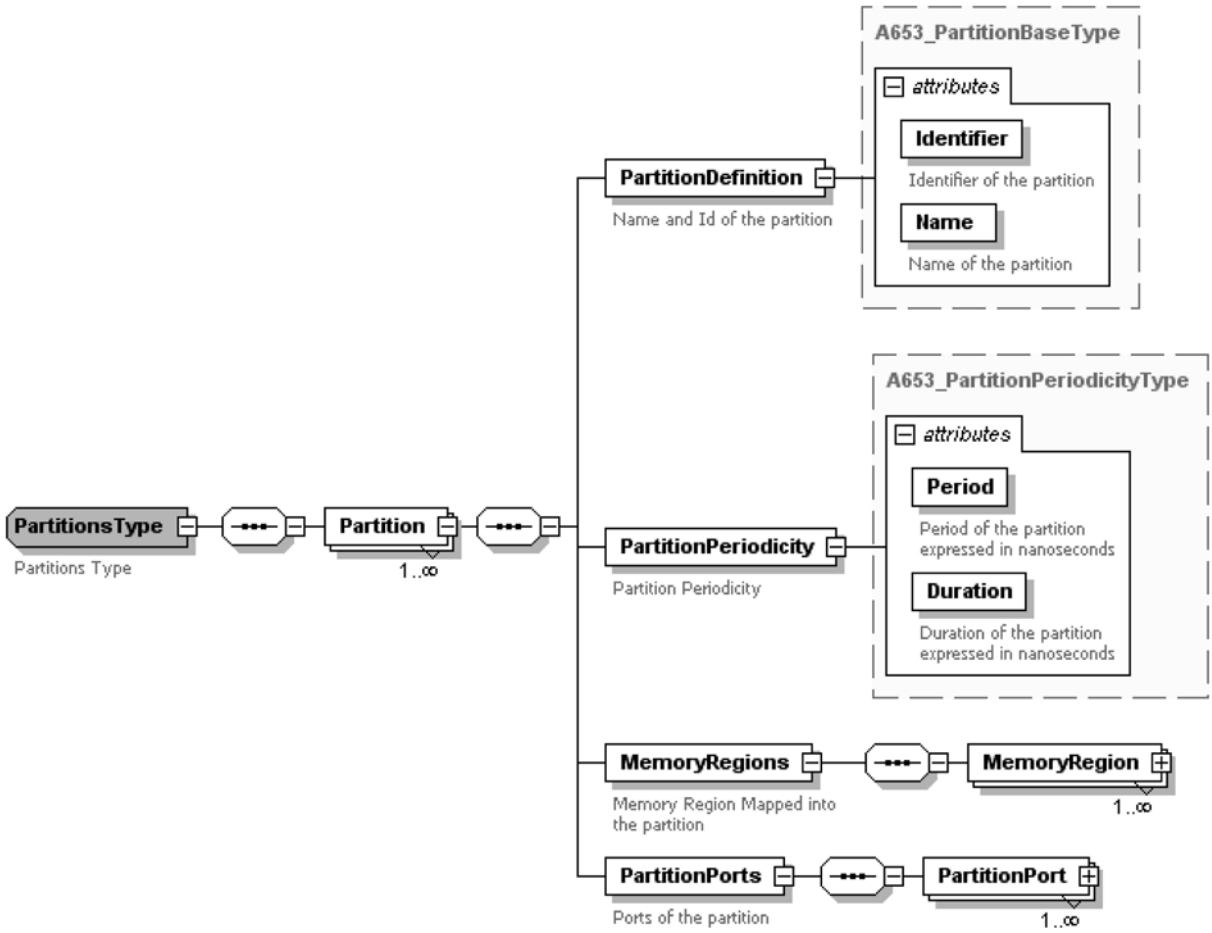


Рис. 3. Описание разделов ([1, с. 200])

Для каждого порта (PartitionPort) задают: имя порта; его тип (QueingPort или SamplingPort); максимальный размер сообщения; направление передачи сообщений в порте (source или destination); для порта с очередью сообщений — максимальное число сообщений в очереди. Начиная с третьей редакции из спецификации исключена таблица соединений, которая раньше определяла соединения портов между собой и представляла собой массив каналов. Для каждого канала указывалось его имя и идентификатор, а также описание порта-отправителя (*source*) и описание порта или портов-получателей (*destination*). Канал с очередью сообщений содержит один порт-отправитель и один порт-получатель, а канал без очереди сообщений содержит один порт-отправитель, а также один или несколько портов-получателей. Если порт принадлежит разделу, для него указывают идентификатор раздела и имя порта. Если порт принадлежит драйверу, его описание содержит имя устройства, идентификатор партнера, номера связи и параметр, передаваемый драйверу устройства. В третьей редакции спецификации в таблицах конфигурации отсутствуют требования к описанию каналов. Считается, что вид и способ конфигурирования каналов определяются разработчиками ОС.

В ОСРВ Багет 3 для описания каналов используется таблица связей, в которой указывается, как порты различных разделов и псевдоразделов связаны посредством каналов. Таблица связей состоит из единственного элемента Connection_Table, который содержит последовательность элементов Channel, по одному на каждый канал. Канал с очередью сообщений содержит один порт-отправитель и один порт-получатель. Канал без очереди сообщений содержит один порт-отправитель и один или несколько портов-получателей. Если порт принадлежит разделу, описание порта состоит из элемента Standard_Partition. В противном случае (если порт принадлежит драйверу устройства) описание порта состоит из элемента Pseudo_Partition.

Для передачи данных между двумя модулями в ОСРВ Багет 3 создаются два канала: один на модуле, передающем данные, другой на модуле, принимающем данные. На модуле, передающем данные, порт-отправитель находится в стандартном разделе, а порт-приемник — в псевдоразделе (*pseudo partition*). На модуле, принимающем данные, порт-приемник находится в стандартном разделе, а порт-передатчик — в псевдоразделе.

Для установления соответствия между каналами, расположенными на разных модулях, согласно спецификации ARINC 653 используют так называемые межмодульные логические связи. Для одной и той же пары модулей может быть установлено несколько логических связей, каждая из которых может использоваться для передачи данных только в одном направлении. Максимальное число межмодульных логических связей указывается при конфигурировании ядра. Логические связи каждой пары модулей нумеруются с 1 (независимо от связей других пар модулей). Реализация конфигурирования каналов в ОСРВ Багет 3 совпадает со второй редакцией спецификации.

Атрибут Schedules (расписания), используемый при описании конфигурации модуля, имеет составной тип, представленный на рис. 4 (см. третью сторону обложки). Как видно на этом рисунке, для описания расписания используется последовательность, состоящая из временных окон для разделов.

Одно окно содержит ссылку на имя раздела, который будет выполняться в этом окне, длительность окна в наносекундах, смещение относительно начала основного периода и, если в разделе, выполняемом в этом окне, содержится периодический процесс, точку старта этого процесса, совпадающую с началом окна для раздела. Длительность основного периода представляет собой наименьшее общее кратное между периодичностями выполнения всех разделов (атрибут Period, см. рис. 2). Такой формат конфигурационной таблицы расписания появился в третьей редакции спецификации, до выхода этой редакции конфигурирование расписаний присутствовало только в расширенном описании спецификации, когда у модуля может быть несколько расписаний, каждое из них идентифицируется своим именем.

В ОСРВ Багет 3 реализовано конфигурирование расписаний, соответствующее расширенной спецификации с небольшими отклонениями, в соответствии с которыми в одном временном окне может выполняться более одного раздела.

Элемент HealthMonitoring (обработка ошибок), представленный на рис. 5 (см. третью сторону обложки), имеет составной тип и содержит массивы элементов: SystemErrors; ModuleHM; MultiPartitionHM; PartitionHM.

Один элемент массива SystemErrors (ошибки системы) содержит описание одной ошибки, состоящее из следующих атрибутов:

- ErrorIdentifier — идентификатор ошибки;
- Description — описание ошибки.

Один элемент массива ModuleHM (ошибки модуля) описывает реакцию на ошибки в зависимости от этапа выполнения системы и содержит:

- атрибут StateIdentifier (идентификатор этапа выполнения);
- атрибут Description (описания этапа выполнения);
- массив элементов ErrorAction (реакция на ошибку), каждый из которых содержит идентификатор ошибки и действие, которое должна предпринять система на эту ошибку; все ошибки считаются ошибками уровня модуля.

Один элемент массива MultiPartitionHM (ошибки разделов) описывает реакцию на ошибки в зависимости от выполняемого раздела и содержит:

- атрибут TableName (имя раздела);
- массив элементов ErrorAction (реакция на ошибку), каждый из которых содержит идентификатор ошибки; уровень ошибки; действие, которое должна предпринять система, если это ошибка уровня модуля.

Один элемент массива PartitionHM (ошибки раздела) описывает реакцию на ошибки для выполняемого раздела и содержит:

- атрибут, содержащий идентификатор элемента массива MultiPartitionHM, который определяет раздел;
- атрибут TableName (имя таблицы);
- массив элементов ErrorAction (реакция на ошибку), каждый из которых содержит идентификатор ошибки, уровень ошибки, действие, которое должна предпринять система, код ошибки, передаваемый обработчику ошибок, если это ошибка уровня процесса.

Реализация этой части спецификации в ОСРВ Багет 3 соответствует предыдущей редакции спецификации, которая несколько отличается составом и наименованием массивов элементов.

В спецификации ARINC 653 не содержится определенных требований к порядку использования таблиц конфигурирования при реализации в конкретной системе. В ОСРВ Багет 3 при реализации ARINC-конфигурирования применялся следующий подход.

Разработана утилита (cm), которая на основе таблиц описания конфигурирования на языке XML-схема из спецификации ARINC 653 формирует графический интерфейс для задания параметров конфигурирования. Синтаксический контроль заданных параметров осуществляется на основе тех же таблиц. Результатом работы конфигуратора является XML-файл, в котором описывается конфигурация конкретного модуля. Этот файл может корректироваться средствами этой же утилиты.

XML-файл конфигурации модуля дополнительно проверяется на логическую непротиворечивость с учетом параметров конфигурации ОСРВ Багет 3. Утилита, осуществляющая эту дополнительную проверку, называется xmltools и может использоваться также для генерации XML-файлов конфигурации модулей многопроцессорных систем.

XML-файл конфигурации модуля транслируется в исходный текст на языке С, в результирующем тексте конструкции языка XML представлены в виде массивов, структур, объединений и простых типов языка С. Далее полученный текст на языке С транслируется С-компилятором [10]; результирующий объектный модуль является входными данными для проверки соответствия образа ОСРВ параметрам, заданным системным интегратором при конфигурировании модуля.

Подмножество интерфейсов

В четвертой части спецификации [5] определены функции, которые можно использовать в рамках подмножества интерфейсов, а также описаны особенности их реализации. Одной из целей разработки этого подмножества было создание более простых и надежных систем. Для этого средства управления процессами были значительно сокращены. В частности, в рамках одного раздела может выполняться не более двух процессов: один периодический и/или один непериодический процесс.

Приоритеты процессам не назначаются, периодический процесс всегда более приоритетный, чем непериодический процесс. Для управления процессами можно использовать функции CREATE_PROCESS() и START(), а также PERIODIC_WAIT() и GET_TIME(). В подмножестве отсутствуют средства взаимодействия процессов внутри одного раздела (семафоры, события, очереди сообщений и доски объявлений). Также отсутствуют функции TIMED_WAIT и REPLENISH(). Существуют лишь две причины, в силу которых процесс может находиться в состоянии ожидания:

- ожидание перехода раздела в рабочий режим;
- ожидание начала следующего периода (для периодических процессов).

Функции передачи данных по каналам упрощены. Они не могут переводить вызывающий их поток в состояние ожидания. Если операция не может быть выполнена, то функция немедленно возвращает управление с соответствующим кодом возврата. Средства обработки ошибок также существенно упрощены. Поток обработки ошибок не поддерживается. Ошибки могут обрабатываться только на уровне раздела или модуля.

* * *

В заключение можно отметить, что спецификация ARINC 653 хорошо подходит для управляющих систем реального времени с повышенными требованиями к обеспечению надежности и безопасности. Подтверждением этого является тот факт, что хорошо известные ОСРВ, используемые в авиации, такие как vxWorks [11], LynxOS-178 [12], Integrity-178B [13], поддерживают спецификацию ARINC 653. Исключение составляет ОСРВ QNX NEUTRINO [14], однако в ней реализована поддержка функционально похожей технологии. Более того, так как спецификация ARINC 653 хорошо известна и понятна международным сертифицирующим органам, то ОСРВ, соответствующая ARINC 653, имеет преимущества при получении сертификата DO-178c [15], необходимого для использования системы в авиации.

Список литературы

1. **ARINC** Specification 653: Avionics Application Software Standard Interface, Part 0 — Overview of ARINC 653. Aeronautical Radio INC, Maryland, USA, 2013.
2. **ARINC** Specification 653-3: Avionics Application Software Standard Interface, Part 1 — Required Services. Aeronautical Radio INC, Maryland, USA, 2010.
3. **ARINC** Specification 653-2: Avionics Application Software Standard Interface, Part 2 — Extended Services. Aeronautical Radio INC, Maryland, USA, 2012.
4. **ARINC** Specification 653: Avionics Application Software Standard Interface, Part 3 — Conformity Test Specification For ARINC 653 Required Services. Aeronautical Radio INC, Maryland, USA, 2014.

5. **ARINC** Specification 653: Avionics Application Software Standard Interface, Part 4 — Subset Services. Aeronautical Radio INC, Maryland, USA, 2012.
6. **ARINC** Specification 653: Avionics Application Software Standard Interface, Part 5 — Core Software Recommended Capabilities. Aeronautical Radio INC, Maryland, USA, 2014.
7. **Годунов А. Н.** Операционная система реального времени Багет 3.0 // Программные продукты и системы. 2010. № 4. С. 15—19.
8. **Годунов А. Н., Солдатов В. А.** Операционные системы семейства Багет (сходство, отличия и перспективы) // Программирование. 2014. № 5. С. 68—76.
9. **IEEE Std** 1003.1, 2004 Edition. The Open Group Technical Standard. Base Specifications, Issue 6. IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002 and IEEE Std 1003.1-2001/Cor 2-2004. System Interfaces.
10. **Build** a GCC-based crosss compiler for Linux. URL: https://www6.software.ibm. com/developerworks/education/l-cross/l-cross-ltr.pdf.
11. **WIND** RIVER VXWORKS 653 PLATFORM. URL: http://www.windriver.com/ products/product-overviews/PO_VE_6_9_Platform_0211.pdf.
12. **LynxOS-178** RTOS for DO-178B Software Certification. URL: http://www.lynx.com/products/real-time-operating-systems/lynxos-178-rtos-for-do-178b-software-certification/
13. **Safety** Critical Products: INTEGRITY®-178B RTOS. URL: http://www.ghs.com/products/safety_critical/integrity-do-178b.html
14. **QNX** Neutrino RTOS 6.6. URL: http://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html
15. **DO-178C** Software Considerations in Airborne Systems and Equipment Certification. URL: http://www.rtca.org/store_product.asp?prodid=803

A. N. Godunov, Head of department, e-mail: nkag@niisi.ras.ru,
V. A. Soldatov, Senior Scientific researcher, e-mail: nkvalera@niisi.ras.ru,
Scientific Research Institute for System Analysis of the Russian Academy of Sciences (SRISA), Moscow

ARINC Specification 653 and its Implementation in RTOS Baget 3

The paper describes the main aspects of the ARINC Specification 653 (Avionics Application Software Standard Interface). All Parts of ARINC Specification 653 are reviewed: Required Services and corresponding XML-configuration specification using this interface; Extended Services; Subset Services; Conformity Test Specification; port device driver services. The distinctions between similar services of POSIX Standard and ARINC specification 653 are noted: area of application; terminology; a number of interface functions. The following special features of ARINC specification 653 are noted: partition modes; resource management and its definition in configuration by system integrator; error handling based on error levels. Partition management based on module scheduling is considered. Module schedules are specified in configuration. All required services of ARINC Specification 653 are considered: interpartition communications (channels and ports management both queuing, sampling and SAP); intrapartition communications (semaphores, blackboards and buffers); logbooks; file systems; error handling and health monitoring; XML configuration specifications. For each service comparison with appropriate POSIX service is given and RTOS Baget 3 implementation of those services are discussed. Well-known RTOS's conforming with ARINC Specification 653 implementation are listed.

Keywords: *ARINC 653, POSIX, real-time operating system, reliability, RTOS Baget 3*

References

1. **ARINC** Specification 653: Avionics Application Software Standard Interface, Part 0 — Overview of ARINC 653. Aeronautical Radio INC, Maryland, USA, 2013.
2. **ARINC** Specification 653-3: Avionics Application Software Standard Interface, Part 1 — Required Services. Aeronautical Radio INC, Maryland, USA, 2010.
3. **ARINC** Specification 653-2: Avionics Application Software Standard Interface, Part 2 — Extended Services. Aeronautical Radio INC, Maryland, USA, 2012.
4. **ARINC** Specification 653: Avionics Application Software Standard Interface, Part 3 — Conformity Test Specification For ARINC 653 Required Services. Aeronautical Radio INC, Maryland, USA, 2014.
5. **ARINC** Specification 653: Avionics Application Software Standard Interface, Part 4 — Subset Services. Aeronautical Radio INC, Maryland, USA, 2012.
6. **ARINC** Specification 653: Avionics Application Software Standard Interface, Part 5 — Core Software Recommended Capabilities. Aeronautical Radio INC, Maryland, USA, 2014.
7. **Godunov A. N.** Operatsionnaya sistema real'nogo vremeni Baget 3.0. *Programmnye produkty i sistemy*, 2010, no. 4, pp. 15—19 (in Russian).
8. **Godunov N. A., Soldatov V. A.** Operatsionnye sistemy semeistva Baget (skhodstvo, otlichiya i perspektivy). *Programmirovanie*, 2014, no. 5, pp. 68—76 (in Russian).
9. **IEEE Std** 1003.1, 2004 Edition. The Open Group Technical Standard. Base Specifications, Issue 6. IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002 and IEEE Std 1003.1-2001/Cor 2-2004. System Interfaces.
10. **Build** a GCC-based crosss compiler for Linux, available at: https://www6.software.ibm. com/developerworks/education/l-cross/l-cross-ltr.pdf.
11. **WIND** RIVER VXWORKS 653 PLATFORM, available at: http://www.windriver.com/products/product-overviews/PO_VE_6_9_Platform_0211.pdf
12. **LynxOS-178** RTOS for DO-178B Software Certification, available at: http://www.lynx.com/products/real-time-operating-systems/lynxos-178-rtos-for-do-178b-software-certification/
13. **Safety** Critical Products: INTEGRITY®-178B RTOS, available at: http://www.ghs.com/products/safety_critical/integrity-do-178b.html
14. **QNX** Neutrino RTOS 6.6, available at: http://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html
15. **DO-178C** Software Considerations in Airborne Systems and Equipment Certification, available at: http://www.rtca.org/store_product.asp?prodid=803

УДК 004.4

В. В. Липаев, д-р техн. наук, проф., гл. науч. сотр., e-mail vlip28@mail.ru,
Институт системного программирования РАН, г. Москва

К разработке моделей динамической внешней среды для испытаний заказных программных продуктов

На основе опыта автора по созданию моделей динамической внешней среды для испытаний критически важных программных управляющих комплексов представлены подходы к разработке требований к моделям такой среды, а также возможное содержание компонентов, генерирующих динамические модели среды, и особенности испытаний заказных программных продуктов во взаимодействии с моделями внешней среды.

Ключевые слова: *заказной программный продукт, внешняя среда управляющих программных продуктов, динамические объекты внешней среды, модели внешней среды, испытания объектов внешней среды*

Введение

При разработке программных комплексов управления сложноорганизованными целевыми системами, а также изменяющимися во времени (динамическими) ***объектами внешней среды***, с которыми подобные системы взаимодействуют, целесообразно выделять требования к комплексу управляющих программ ***и требования к модели, имитирующей внешнюю среду (ВС) подлежащего созданию программного продукта***. Такие сложные, наукоемкие программные продукты, как правило, создают по заказу промышленных предприятий и организаций. Заинтересованное участие заказчиков и разработчиков вновь создаваемого программного комплекса в формировании таких требований необходимо для того, чтобы обеспечить должную полноту и точность набора требований, позволяющих адекватно ***описать характеристики разрабатываемого оборудования и процессов его взаимодействия с объектами внешней среды.*** Как следствие, при определенном исходном состоянии подлежащего разработке программного продукта и множестве входных данных разработчики должны ***предсказать состав и содержание выходных, близких к прогнозируемым данных управляющего программного продукта и всей динамически изменяющейся целевой системы, взаимодействующей с моделями ВС***.

Имитация конкретных динамических объектов с реальными характеристиками, адекватными характеристикам объектов ВС, является основной частью моделирующих имитационных стендов (см. рисунок). Качество моделей зависит от объема и глубины знаний разработчиков алгоритмов функционирования внешних объектов, характеристик их компонентов и обобщенных параметров динамических режимов функционирования целевой системы в целом. Такими динамическими объектами во внешней среде могут быть: динамические объекты воздушного, наземного и водного видов транспорта; операторы-пользователи объектов ВС; результаты анализа состояния аппаратуры объектов ВС; тренажеры пользователей динамических объектов ВС.

Все эти объекты находятся под управлением программных продуктов с использованием обратной связи, позволяющей получить информацию о состоянии и изменениях объектов ВС.

Отдельного внимания в процессе разработки управляющих программных продуктов заслуживают вопросы оценки их качества. С этих позиций необходимо, чтобы ***имитаторы динамической внеш-***

Схема функционирования в динамике программных продуктов с использованием объектов внешней среды

ней среды [1] создавали условия эксплуатации таких продуктов, близкие к реальным. Такая адекватность зависит от степени учета основных и второстепенных факторов, характеризующих функционирование реальных объектов и источников информации. Точность требований к моделям ВС и их программной реализации прежде всего определяется алгоритмами, на которых они базируются, а также полнотой учета в них всех особенностей функционирования моделируемых объектов ВС. Кроме того, отметим, что на корректность имитации ВС влияет уровень дефектов и ошибок, допущенных в реализующих ее программах. Каждый неучитываемый в имитаторе элемент или фактор моделируемой системы необходимо оценивать путем сопоставления характеристик частных имитируемых данных с результатами аналитических исследований или с данными, полученными на реально работающих системах. Следует также определять возможное влияние такого фактора на формулируемые к модели ВС требования, а также на точность описания модели и генерируемых ею динамических свойств. При этом нужно принимать во внимание другие составляющие, которые влияют на достоверность результатов имитационного моделирования.

Разумное ***сочетание части реальных (физических) объектов ВС*** (когда это возможно) ***и объектов, которые моделируются на компьютерах,*** может обеспечивать создание высокоэффективных моделирующих испытательных стендов (МИС). Такие комплексные (гибридные) модели крайне необходимы для динамических испытаний качества программных продуктов, управляющих динамическими системами реального времени. Важнейшая задача формирования требований к модели ВС состоит в достижении основной цели всей системы управления, в которой применяется заказной программный продукт. Моделирующие испытательные стенды позволяют осуществлять автоматизированную генерацию тестов с помощью имитаторов на ***компьютерах*** и аналогов реальной аппаратуры. Они представляют возможности дополнять создаваемый программный продукт и систему в целом реальными данными от операторов-пользователей, контролирующих и корректирующих процесс функционирования систем обработки информации. Также модель ВС должна предоставлять возможность разработчику и заказчику определять характеристики качества при принятии решения о ***готовности программного продукта для использования по назначению***. Автоматизация испытаний с использованием модели ВС способствует более быстрому, качественному и эффективному тестированию создаваемого программного комплекса. Это обстоятельство, в свою очередь, ведет к сокращению объема работ и к улучшению процесса производства программного продукта. В ходе испытаний должны проверяться условия того, что комплекс программ работает в соответствии с предъявляемыми к нему требованиями и ***удовлетворяет заданным критериям качества всей целевой системы.***

Разработка требований к модели динамической внешней среды

Процесс разработки модели ВС для управляющего программного комплекса реального времени целесообразно начинать с определения требований к такой модели. Далее следует верификация требований к отдельным компонентам и к модели в целом. Эти требования к модели ВС должны соответствовать требованиям к подлежащему созданию программному продукту. Для заказных комплексов программ важно иметь корректные ***требования к такому продукту и сценарии его использования, а также модели ВС для управляющего программного комплекса***. Для этого при анализе требований к модели ВС необходимо [2]:

- изучить требования к целевой системе и к управляющему программному продукту как его составляющей, назначение и сценарии использования создаваемого программного продукта и внешней среды для того, чтобы более точно формализовать эти требования;
- определить наиболее значимые для программного продукта функции, а также функции, реализация которых связана с повышенным риском неблагоприятных событий;
- определить требования к тестовым испытаниям для проверки корректности версии управляющего программного продукта и компонентов внешней среды;
- преобразовать архитектуру, требования к функциям и характеристикам программного продукта в сценарии и результаты его использования;
- по результатам анализа требований к системе и к результатам функционирования модели ВС сформировать сопроводительную документацию на комплекс программ.

Требования к модели ВС должны содержать подробный перечень тех сущностей (объектов, их атрибутов, свойств и т. п.), которые должны функционировать и подлежать испытаниям. Цели верификации модели на предмет соответствия составляющих ее сущностей предъявляемым к ним требованиям достигаются путем анализа потенциально возможных ситуаций, разработкой контрольных сценариев и процедур, а также возможных результатов последующего их выполнения. Такие сценарии, по результатам которых анализируются характеристики ***модели ВС, предназначены для поэтапной проверки внутренней непротиворечивости и полноты реализации совокупности требований к модели ВС.*** Однако следует иметь в виду, что результаты такого анализа не гарантируют полноту и корректность всех требований. Причина в том, что эти работы проводятся частично вручную и, как следствие, могут отсутствовать четкие "эталоны" для их проверки.

Для обеспечения высокого качества управляющего программного продукта одновременно с ***верификацией требований*** и с их модификацией следует разрабатывать и верифицировать ***сценарии***, отражающие методы

и конкретные процедуры проверки выполнения этих требований. Специфицированные требования могут использоваться для проверки согласованности, внутренней непротиворечивости и степени полноты выполнения требований без исполнения программ. Для каждого требования к функциям комплекса программ, к его архитектуре, к отдельным компонентам должна быть разработана спецификация требований к тестам, обеспечивающая проверку их корректности, адекватности реалиям и возможности в последующем реализовать тестирование компонентов на соответствие такому требованию. Такая взаимная проверка корректировок функций компонентов модели ВС, отраженных в требованиях и в спецификациях тестов, обеспечивает повышение их качества, сокращение дефектов, ошибок, неоднозначностей и противоречий.

Наборы требований к модели ВС могут применяться ***как "эталоны"*** и другая адекватная форма описания функций и других характеристик комплексов управляющих программ. Для обеспечения высокого качества программных продуктов стратегию создания ***требований к модели ВС*** целесообразно строить***, основываясь на требованиях к функциям и характеристикам целевой системы***. Требования к модели могут пополняться на основе анализа логики функционирования и архитектуры комплекса программ путем построения структуры решений. Такой подход может применяться в зависимости от условий договора исполнителя работ с заказчиком или особых требований к безопасности программного комплекса. Например, покрытие тестами полной структуры программных решений для некоторых программных комплексов может быть необходимо в аэрокосмической отрасли и в других критически важных динамических системах с повышенными требованиями к их безопасности.

Компоненты генераторов динамических моделей внешней среды

Для проведения динамических испытаний комплексов программ реального времени следует составлять планы автономных сценариев функционирования модели ВС и готовить ***обобщенные исходные данные.*** Программы моделирования должны обеспечивать реализацию этих сценариев на основе конкретных значений данных (параметров), характеризующих динамические режимы функционирования каждого имитатора или реального (физического) ***объекта внешней среды***. Эти данные могут вводиться и преобразовываться на моделирующем компьютере как в режиме offline, на этапе подготовки к функционированию МИС, так и в ходе испытаний программы управления объектами ВС в реальном времени (online).

Аналоги компонентов функционирования системы и программные модели ВС используются для генерации динамических тестов. Кроме того, они позволяют проверять и аттестовывать некоторые программные и аппаратные имитаторы объектов внешней среды, которые впоследствии используются при испытаниях.

Данные с рабочих мест операторов-пользователей управляющей системы должны имитировать реальные характеристики различного рода динамических воздействий на программный продукт. При этом необходимо учитывать особенности и квалификацию администратора (оператора), которому предстоит использовать программы как в реальной системе управления, так и в моделях ВС. На эту часть МИС, кроме первичных исходных данных от операторов, могут вводиться данные, полученные в результате обработки некоторых динамических тестов системы, которая включает и управляющие программы, и модели ВС.

Данные натурных экспериментов с объектами ВС могут готовиться и фиксироваться заранее, вне сеансов динамических испытаний программного продукта, например при отладке аппаратной части моделируемой системы обработки информации. Эти данные могут отражать характеристики и динамику функционирования моделируемых объектов, которые трудно или опасно подключать для непосредственного взаимодействия с недостаточно проверенными управляющими программами. Кроме того, такие данные могут использоваться для проверки адекватности разрабатываемых имитаторов некоторых объектов ВС реально существующим объектам.

При динамическом моделировании в ряде случаев необходимо иметь ***эталонные (соответствующие реальным) характеристики***, которые поступают на подлежащий испытаниям управляющий программный продукт или систему в целом. При работе с реальными объектами зачастую приходится создавать специальные измерительные комплексы для получения таких характеристик в процессе наблюдения за реальным функционированием динамических объектов. К числу таких характеристик относятся, например, динамические характеристики и координаты движения самолетов при испытаниях систем управления воздушным движением. Подобные измерения, как правило, проводятся в режиме автономного функционирования внешних объектов, или подлежащие определению исходные данные или характеристики регистрируются при взаимодействии таких объектов с управляющим комплексом программ в реальном времени.

Повторяемость сеансов испытаний при автоматической имитации динамических объектов ВС должна обеспечиваться фиксацией всех исходных данных и применением программного формирования псевдослучайных чисел (если они используются). При надежной работе аналогов реальных объектов и компьютера, моделирующего ВС, можно добиться высокой степени повторяемости подлежащих измерению характеристик в ходе длительных экспериментов и сценариев моделирования. Труднее обеспечивать такую повторяемость в ходе реализации сценариев динамических испытаний, в которых активно участвует оператор-администратор. В этом случае необходимо регистрировать и сохранять действия оператора в зависимости от времени, а затем повторять их в соответствии с принятым сценарием. При необходимости

временная диаграмма может соблюдаться с точностью до 0,5...1 с. Однако ошибки в действиях оператора и, соответственно, во вводимых им параметрах могут отличаться в каждом сценарии моделирования.

После приемки заказчиком или пользователями в процессе функционирования всей системы и применения управляющего программного продукта должна обеспечиваться оценка его текущего качества с моделями ВС. Для этого в ***составе программного продукта необходимо иметь средства, обеспечивающие:***

- генерацию динамических сценариев или хранение состояний моделей ВС для контроля работоспособности, сохранности и целостности управляющего программного продукта;
- оперативный контроль и обнаружение дефектов исполнения моделирующих программ и механизмов обработки данных при использовании программного продукта по прямому назначению;
- реализацию процедур предварительного анализа выявленных дефектов моделей ВС и оперативное восстановление вычислительного процесса, программ и данных (рестарт) после обнаружения аномалий моделей динамического функционирования программного продукта или всей системы;
- мониторинг, накопление и хранение данных о выявленных дефектах, сбоях и отказах в процессе функционирования управляющего комплекса программ и в ходе обработки данных с использованием моделей ВС.

Средства генерации динамических тестов и имитации ВС совместно с поставляемым заказчику программным продуктом могут использоваться для оперативной подготовки исходных данных при проверке различных режимов функционирования программного продукта, а также при диагностике проявившихся дефектов. Минимальный состав средств генерации моделей ВС должен передаваться пользователям для контроля использования рабочих версий в реальном времени. Он может входить в комплект поставки каждой пользовательской версии управляющего программного продукта.

Важной функцией испытательных стендов ВС является их использование в качестве ***тренажеров для операторов-пользователей***. Качество функционирования заказных комплексов программ может существенно зависеть от характеристик конкретных операторов, участвующих в эксплуатации и обработке информации. Как следствие возникает потребность в определении этих характеристик. Более того, необходимо иметь возможность улучшать их до уровня, который обеспечивает выполнение ***заданных требований к управляющему программному продукту***. По этой причине в программу испытаний в обязательном порядке должны входить процедуры динамической тренировки операторов, измерения реальных характеристик функционирования программного продукта и времени реакции оператора на внешние воздействия. Важным направлением является использование МИС для обучения и регулярной подготовки операторов-пользователей в процессе тиражирования и эксплуатации программного продукта и его моделей ВС.

Испытания программных продуктов с динамическими моделями внешней среды

До проведения тестовых испытаний разрабатываемого программного продукта заказчик и исполнитель должны подготовить, согласовать и ***утвердить план его испытаний с учетом взаимодействия с ВС.*** С этой целью следует описать требования к таким испытаниям. Они должны ***соответствовать требованиям, которые предъявляются к системе в целом, включающей программный комплекс и объекты ВС, а также к тестовым сценариям.***

Испытания целесообразно делить на следующие три этапа:

1) автономные испытания управляющего заказного программного продукта и его воздействий на объекты ВС;

2) автономные испытания динамических объектов и других компонентов моделей ВС;

3) комплексные испытания управляющей системы с динамическими объектами и моделями ВС.

Первый этап представляет собой традиционные испытания сложных программных продуктов, которые дополнены требованиями проверки наличия полноты и корректности (соответствия реально существующим) воздействий разрабатываемого программного комплекса на объекты ВС, а также механизмов обработки поступающей от этих объектов информации. На этом этапе должен учитываться план автономных динамических испытаний объектов и других компонентов модели ВС. Следует также оценить ресурсозатраты на испытания управляющего программного продукта с динамической моделью ВС. При этом необходимо принимать во внимание следующие обстоятельства:

- испытания в первую очередь целесообразно проводить для проверки выполнения требований с наивысшим приоритетом, которые наиболее важны для заказчика и пользователей, либо тех, невыполнение которых может привести к значительному ущербу для качества системы;
- сначала следует осуществлять испытания новых функциональных возможностей, направленных на исправление или совершенствование характеристик управляющего программного продукта;
- тестирование полезно начинать с функций, с которыми наиболее часто будет иметь дело конечный пользователь или лица, имеющие отношение к взаимодействию с объектами системы в целом.

Второй этап должен начинаться с выбора типов объектов и других компонентов для построения моделей ВС. Их целесообразно упорядочивать по приоритетам, по сложности и важности. Эти факторы принимаются во внимание при определении последовательности их разработки, проведения испытаний и использования. К числу таких объектов и компонентов относят:

- программные модули;
- аппаратные средства;
- воздействия от операторов-пользователей;
- результаты взаимодействия ВС с управляющим комплексом, полученные в ходе предварительных натурных экспериментов;
- тренажеры.

При этом наиболее гибкие программные компоненты комплекса могут приспосабливать свой интерфейс к объектам и компонентам ВС, которые изменять намного сложнее.

Третий этап — испытания всей системы, которая включает заказной программный продукт, с комплексом имитационных динамических моделей ВС. Результаты испытаний на этом этапе призваны продемонстрировать заказчику выполнение всех согласованных с ним требований к проверяемой динамической системе. Планирование испытаний включает как определение требований к тестам, так и разработку процессов управления этими требованиями.

Когда план испытаний комплекса управляющих программ разработан и полностью описывает процессы тестирования, он становится руководящим документом (базовым инструментарием) для формирования ***программы динамических испытаний системы в целом***. Эта программа может уточняться посредством корректировок целей, задач и стратегий тестирования отдельных моделей объектов и других компонентов ВС в реальном времени. Эти объекты и компоненты могут быть не только программными, но и представлять собой аппаратные средства или воздействия от операторов.

Для сокращения неопределенностей и прямых ошибок при оценивании качества программного продукта необходимо до начала испытаний определить основные параметры ВС, при которых должен функционировать комплекс программ с требуемыми характеристиками. Для этого заказчик и разработчик совместно должны структурировать, описать и согласовать ***модель внешней среды*** и ее параметры в среднем, типовом режиме применения программного продукта, а также в наиболее вероятных и критических режимах, в которых должны обеспечиваться требуемые характеристики качества функционирования всей системы.

Программа испытаний является планом проведения серии экспериментов и должна разрабатываться с позиции допустимой минимизации объема тестирования в процессе проведения испытаний. Программа испытаний, методики их проведения и оценки результатов разрабатываются совместно заказчиком и исполнителем. Программа содержит уточнения и детализацию требований для данного программного продукта. Ее задача — гарантировать корректную проверку всех заданных характеристик качества функционирования объектов ВС.

До начала испытаний программного продукта подлежат проверке и паспортизации средства, которые обеспечивают: получение эталонных данных; имитацию тестов от внешних объектов; фиксацию и обработку результатов тестирования ВС. При подобных испытаниях важную роль играет оценка и обеспечение близких значений ожидаемых по методике и статистической достоверности результатов испытаний. Завершение испытания программного продукта с моделями ВС должно сопровождаться предъявлением заказчику на утверждение ***комплекта документов, содержащих результаты комплексных испытаний*** создаваемых программных продуктов и системы в целом, для которых они создаются.

При разработке сложных комплексов программ для их верификации и тестирования, как правило, необходимы ***значительные ресурсы*** в течение всего их жизненного цикла. Наиболее критичным ресурсом при этом является ***допустимое время*** поэтапного выполнения этих процедур. В результате совокупность ***требований к заказному программному продукту*** может применяться ***как эталон и дополнительная адекватная форма описания содержания комплекса управляющих программ и системы, для которой этот комплекс создается.*** Такие ***параллельные взаимные проверки*** требований, текстов управляющих программ и моделей ВС способствуют выявлению и исключению вторичных дефектов и ошибок комплексов программ. В дальнейшем эти требования должны использоваться для тестирования выполнения требований к программным компонентам создаваемого комплекса и модели ВС. Параллельная и независимая разработка управляющих программ и моделей ВС, а также их реализация позволяют распараллеливать эти независимые во времени сферы деятельности разработчиков. Это обстоятельство ведет к сокращению сроков создания компонентов и комплексов программ.

Особенности описаний и реализации программ, которые проявляются при использовании требований, функций, характеристик структуры и исполнения управляющих программ, ***существенно отличаются от представлений и методов описаний таких же функций комплекса программ ВС***. Создатели сценариев тестирования и эталонов акцентируют свою деятельность на конкретных процедурах проверки функционирования, на возможных результатах и взаимодействии объектов ВС. ***Такой подход позволяет выявлять вторичные дефекты, повышать качество путем сопоставления двух методов и результатов описания одних и тех же функций и характеристик системы.*** Результат при этом достигается за счет того, что мала вероятность одинаковых ошибок в сценариях и реализациях моделей ВС и в описаниях требований к функциям и к характеристикам управляющих программ.

Важным аспектом организации испытаний заказных программных продуктов и моделей динамических объектов ВС является принятие решения о том, в каком объеме они достаточны и ***когда необходимо завершить процесс тестирования*** [3, 4]. Принятие решения об окончании тестирования включает рассмотрение вопросов стоимости и рисков неблагоприятных ситуаций, связанных с возможными нарушениями надежности функционирования тестируемой программной системы и динамических моделей ВС. В то же время необходимо отметить, что ресурсозатраты на тестирование также являются одним из ограничений, на основе которых принимается решение о продолжении работ или об их прекращении.

Заключение

Затраты на создание моделей имитации динамических систем ВС для оценивания качества динамических программных продуктов могут быть одной из существенных составляющих при создании крупных

заказных программных комплексов реального времени [5]. В ряде случаев они соизмеримы с затратами на создание основных функций управляющих комплексов программ. Это обстоятельство определяется принципиальным соответствием ***сложности необходимых наборов тестов*** и ***сложности функций***, реализуемых испытываемым комплексом программ.

Автор благодарит д-ра физ.-мат. наук, проф. В. А. Васенина за ценные замечания и большой труд по редактированию этой статьи.

Список литературы

1. **Шеннон Р.** Имитационное моделирование систем — искусство и наука. Пер. с англ. М.: МИР, 1978. 424 с.
2. **Автоматизированные** системы управления воздушным движением. Учеб. пособие / под научной редакцией Ю. Г. Шатракова. СПб.: Политехнология, 2014. 450 с.
3. **Дастин Э., Рэшка Д., Пол Д.** Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация. Пер. с англ. М.: ЛОРИ, 2003. 567 с.
4. **Блэк Р.** Ключевые процессы тестирования. Пер. с англ. М.: ЛОРИ, 2006. 544 с.
5. **Липаев В. В.** Методы обеспечения качества крупномасштабных программных средств. М.: РФФИ. СИНТЕГ, 2003. 520 с.

V. V. Lipaev, Professor, Senior Scientific Researcher, e-mail: vlip28@mail.ru,
Institute for System Programming of the Russian Academy of Sciences, Moscow

To the Development of Dynamic External Environment for Custom Software Tests

*Some software is designed to **control dynamic (changing their state in time) objects.** Such objects can be placed in the environment which is external to the main software of the system (and this software is specifically designed to control this system). Software packages like that can be divided into two actively interacting parts — the **control** part of the target system and **dynamic part which implements the dynamic objects management in external environment.** Different models like spacecraft flight models; models of air-traffic control stations in and around airport; models of objects of antiaircraft defense systems; avionics models — all these models can be external environment objects which are used during development and testing of intricately organized software packages as well as the whole target system. In a process of such software packages development one of the tasks is the creation and testing of mentioned before models of dynamic objects management in external environment. Due to the complexity of full-scale modelling of such objects' dynamics in actual practice, software simulators are used during the design and production stages. Modeling test desks are created for these purposes. They [desks] include problem-oriented complexes of the programs which model objects in the dynamic external environment. It should be noted that such complexes can be much bigger than corresponding tested control software. The first domestic (Soviet) software simulators of external environment, imitating flights of the different types of the aircrafts in external environment, were created in 1960s for testing the country's antiaircraft defense systems.*

The control systems in life cycle of the spacecrafts' onboard complexes of the control programs as well as the aircrafts' flight control systems and the dispatchers' control systems in air traffic control centers can be considered as examples of external environment models to check if they meet the feature and characteristic requirements of the complexes of control programs. The simulation of real-time changes of all information coming from the external environment must be provided for complex debugging and testing of the software of such control systems. The sources of information for the ATC centers' modeling test desks were radar stations, air crews on board, flying control officers. As a result, the necessity of dynamic simulation of several heterogeneous objects interaction, taking into account their influence on the control object, has emerged.

Based on the author's experience in creation of dynamic external environment models for testing of the complexes of the target system control programs, the article presents: the approach to such environment models requirements development; the components which generate dynamic models of the environment; features of custom software testing in interaction with external environment models.

Keywords: *custom software, external environment of manager programs, dynamic objects of external environment, external environment models, external environment objects testing*

References

1. **Shannon R.** *Imitatsionnoe modelirovanie sistem — iskusstvo i nauka* (Systems simulation. The art and science). Moscow: MIR, 1978. 424 p. (in Russian).
2. ***Avtomatizirovannye** sistemy upravleniya vozdushnym dvizheniem*. Uch. posobie (Automated systems of air traffic control) / Eds by Yu. G. Shatrakova. Sankt-Petersburg: Politekhnologiya, 2014. 450 p. (in Russian).
3. **Dusting E., Reshka D., Paul D.** *Avtomatizirovannoe testirovanie programmnogo obespecheniya. Vnedrenie, upravlenie i ekspluatatsiya* (Automated Software Testing. introduction, Management and Performance). Moscow: LORI, 2003. 567 p. (in Russian).
4. **Black R.** *Klyuchevye protsessy testirovaniya* (Key testing processes). Moscow: LORI, 2006, 544 p. (in Russian).
5. **Lipaev V. V.** *Metody obespecheniya kachestva krupnomasshtabnykh programmnykh sredstv* (Methods of ensuring the quality of large-scale software systems). Moscow: SINTEG, 2003. 520 p. (in Russian).

УДК 004.423.4

В. А. Васенин, д-р физ.-мат. наук, зав. лаб., e-mail: vasenin@msu.ru,
М. А. Кривчиков, науч. сотр., e-mail: maxim.krivchikov@gmail.com,
НИИ механики МГУ имени М. В. Ломоносова, г. Москва

Формальные модели программ и языков программирования. Часть 2. Современное состояние исследований

Настоящая статья продолжает опубликованные в № 5, 2015, результаты исследований. На основе анализа публикаций зарубежных и российских авторов с 1990-х годов по настоящее время дана оценка современному состоянию исследований в области формальной верификации программ. Основное внимание при этом уделено подходу к верификации на основе языков программирования с зависимыми типами. Рассмотрены также результаты и на других, смежных направлениях исследований. Обозначены потенциально перспективные, по мнению авторов, направления дальнейших исследований в этой области.

***Ключевые слова*:** *формальная верификация, языки программирования, предметно-ориентированные языки, формальная семантика программ, программная инженерия, библиография*

Введение

В рамках современных подходов к инженерии программ используют несколько различных способов их верификации, а именно визуальный контроль, тестирование, статический анализ, формальную верификацию. Из числа указанных способов только использование формальной верификации позволяет получить строгие математические гарантии корректности программы с позиции ее соответствия некоторой спецификации. Как было отмечено в части 1 настоящей статьи, с 1960-х гг. и по настоящее время научными организациями и коммерческими компаниями по всему миру проводятся исследования, целью которых является получение математических доказательств корректности программного обеспечения. Однако в сложившейся практике разработки программного обеспечения этот способ нельзя отнести к числу широко используемых.

Подходы к формальной верификации в литературе условно делят на два класса — теоретико-модельные и теоретико-доказательные (дедуктивные). В случае теоретико-модельных методов программа описывается в терминах некоторой модели, допускающей исчерпывающую проверку состояний на предмет выполнения требуемых свойств (*model checking* [1]). Для теоретико-доказательных (дедуктивных) методов доказательство выполнения свойств получают как вывод в некоторой формальной системе, в которой задана формальная семантика программы. В последние годы в рамках дедуктивной верификации сформировалось отдельное направление — программирование с зависимыми типами, основанное на наблюдении, получившем название "соответствие Карри—Говарда" [2, 3]. Согласно этому наблюдению доказательства, полученные методом естественной дедукции в его интуиционистской разновидности, могут быть интерпретированы в терминах типизированного λ-исчисления. Таким образом, на базе типизированного λ-исчисления могут быть построены среды автоматизации математических доказательств, язык формальной спецификации которых в то же время является статически типизированным функциональным языком программирования. Формальная семантика языков программирования как общего назначения, так и предметно-ориентированных в рамках таких сред может быть представлена в виде интерпретатора. Последнее обстоятельство может существенно упростить задачу получения формальной модели программы и дальнейшей работы с ней.

В настоящей статье рассматриваются предыстория с 1990-х гг. и современное состояние исследований в области формальной верификации, формальных моделей программ и разработки верифицируемого программного обеспечения. В первую очередь изложение сосредоточено на работах, которые имеют прямое отношение к развитию подходов на основе формальной верификации с использованием программирования с зависимыми типами. При этом также упоминаются связанные с ними фундаментальные работы, результаты которых важны для области формальной верификации программ в целом. Следует отметить, что результаты, относящиеся к методам *model checking*, выходят за рамки настоящей статьи. В связи с высокой популярностью методов *model checking* относительно других подходов к формальной верификации характеристика современного состояния исследований на этом направлении потре-

бовала бы отдельной статьи. На основе приведенных публикаций в заключении представлены примеры возможных направлений будущих исследований по развитию верификации с использованием языков программирования с зависимыми типами.

1. Программное обеспечение как отрасль экономики (1990-е гг.)

Начиная с 1990-х гг. в связи с высокими темпами развития вычислительной техники и программного обеспечения становится затруднительно выделять те или иные основные тенденции такого развития. В связи с этим результаты исследований, представленных в настоящем разделе, целесообразно рассматривать на фоне следующего тренда: выделение предприятий, осуществляющих разработку, интеграцию и сопровождение программного обеспечения, распространяемого отдельно от аппаратно-программной платформы, как крупной отдельной отрасли экономики. Кроме того, во второй половине 1990-х гг. активное развитие, особенно в прикладном плане, получила глобальная сеть Интернет, началось распространение мобильных устройств, постоянно подключенных к телекоммуникационной сети. С позиций инженерии программ выделение разработки программного обеспечения (ПО) в качестве отдельной отрасли привело к распространению методов и средств массовой (промышленной) разработки ПО. Одним из наиболее ярких примеров в этом отношении являются шаблоны проектирования ПО [4].

Язык программирования и виртуальная машина Java [5] представляют собой первый пример широкого промышленного применения языка программирования, обладающего следующими характеристиками: исключительно автоматическое управление памятью; виртуальная машина, использующая статически типизированный байт-код. В дальнейшем в число широко используемых на практике языков программирования с автоматическим управлением памятью войдут такие языки, как C# (во многом повторяющий язык Java, однако использующий другие инженерные решения в некоторых аспектах реализации), а также динамические языки программирования, такие как Lua, Python и ECMAScript. Одним из приложений виртуальной машины Java является Java Card [6] — технология, позволяющая использовать виртуальную машину Java в программном обеспечении смарт-карт (в том числе в настоящее время — SIM-карт). Для подмножества языка Java Card была построена модель формальной семантики [7], с использованием которой предполагалось проводить верификацию ПО с высокими требованиями к безопасности. Эту модель можно отнести в первую очередь к операционному подходу к описанию семантики. Известны также результаты по описанию аксиоматической семантики языка Java [8].

В числе значимых результатов в области формальной верификации, опубликованных в 1990-е гг., следует отметить следующие: разработку известного средства верификации PVS (*Prototype Verification System*) [9]; разработку средств верификации для языка программирования Ada [10]; практическое применение методов формальной верификации для синхронизации часов в цифровых системах авионики [11]; оптимизацию методов *model checking* для существенного снижения объема рассматриваемых состояний при автоматической верификации на некоторых моделях [12]. Обзор исследований по приложению методов формальной верификации к анализу криптографических протоколов изложен в работе [13].

Подход, получивший название языково-ориентированного программирования (*Language-Oriented Programming*), рассмотрен в работе [14]. Следует отметить, что существует большое число ранних примеров использования языков, адекватно отражающих специфику предметной области (предметно-ориентированных языков, DSL; подробнее — в работе [15]). Однако в работе [14] разработка предметно-ориентированных языков рассмотрена как центральный этап разработки крупных программных комплексов: на ранних этапах разработки таких комплексов должны быть разработаны сверхвысокоуровневые, формально специфицированные предметно-ориентированные языки программирования, с использованием которых возможно полное высокоуровневое описание разрабатываемого комплекса. В дальнейшем разработка разделяется на две независимые ветви — реализация комплекса на предметно-ориентированных языках и реализация компиляторов или интерпретаторов таких языков. Следует отметить, что в цитируемом исследовании не рассматривались вопросы формальной верификации, а требование наличия формальной спецификации у предметно-ориентированных языков связывалось исключительно с возможностью независимой разработки непосредственно самой системы и инструментальных средств.

Один из полученных в 1990-х гг. результатов в области формальной семантики языков программирования, который следует отметить особо, — это модель денотационной формальной семантики подмножества языка C (стандарта ANSI C) [16]. В этой работе представлена модель формальной семантики существенного подмножества языка программирования C, используемого на практике. Основная структура модели опирается на предложенный в работе [17] подход к описанию вычислений с использованием монад — специальных конструкций из области теории категорий.

Вопросы применимости формальной верификации в условиях массового распространения программ были рассмотрены в работе [18], где был предложен подход к распространению кода программы под названием *proof-carrying code* (код, несущий доказательство). В рамках такого подхода совместно с исходным или исполнимым кодом программы распространяется сертификат — доказательство выполнения кодом предъявляемых к нему требований. Сертификат представлен в машиночитаемом виде,

что позволяет в дополнение к предоставляемым криптографическими средствами гарантиям подлинности получить и строгие математические гарантии выполнения требований. В целях снижения объема доверенного кода, выполняющего проверку таких сертификатов, позднее в работе [19] был предложен модифицированный вариант подхода — *foundational proof-carrying code* (фундаментальный код, несущий доказательство). В такой модификации сертификат представляет собой вывод в некоторой формальной системе (в оригинальном исследовании была использована эдинбургская логическая система LF). Выбранная формальная система должна быть достаточно выразительна, поэтому в качестве таких систем рассматривают в первую очередь логики высших порядков и теорию типов.

В рамках исследований по развитию исчисления конструкций следует отметить перечисленные далее работы. Расширение исчисления конструкций индуктивными типами [20, 21] позволило рассматривать в рамках этой формальной системы сложные объекты, такие как списки или деревья, а также определять функции, оперирующие такими объектами, без расширения набора аксиом. Независимая формализация индуктивных типов представлена в работе [22]. Модификация, позволяющая интерпретировать ограничения на индуктивные типы непосредственно в рамках формальной системы логики второго порядка, представлена в работе [23]. Дополнение индуктивных типов дуальными им коиндуктивными типами допускает описание в терминах исчисления конструкций моделей сервисного ПО, обрабатывающего потенциально бесконечный поток входных данных (запросов) [24].

На основе исчисления конструкций была разработана система автоматизации и поддержки доказательств Coq, одно из первых упоминаний о которой в литературе содержит работа [25]. Эта система по настоящее время остается основной программной реализацией исчисления конструкций.

2. Распределенные гетерогенные системы (с 2000 г. по настоящее время)

Одна из основных современных тенденций в области вычислительной техники, на фоне которой в настоящем разделе рассмотрены результаты современных исследований, — это распространение параллельных и распределенных вычислительных систем, в том числе гетерогенных. Использование параллельных вычислений в настоящее время необходимо для эффективного использования возможностей широкого класса аппаратных платформ — начиная от высокопроизводительных суперкомпьютеров и заканчивая мобильными устройствами. Интеграция таких, а также более ограниченных по возможностям встраиваемых платформ на настоящее время является одной из основных областей коммерческой разработки ПО. Параллельность, распределенность и гетерогенность при этом добавляют дополнительный уровень сложности в крупные программные комплексы. Требования к безопасности таких комплексов значительно повышаются, поскольку подобные распределенные системы используют, в том числе, и для обеспечения работоспособности критически важных объектов национальных инфраструктур [26, 27].

Современные исследования в области развития инженерии программ выражаются в появлении в 2009 г. инициативы SEMAT [28, 29] (*Software Engineering Method and Theory*, метод и теория программной инженерии), ведущим пропонентом (пропагандистом) которой является И. Якобсон. Целью этой инициативы является разработка новых оснований программной инженерии, позволяющих рассматривать ее как строгую научную дисциплину. На настоящее время основным результатом инициативы является разработка и принятие в 2014 г. стандарта "Essence — Kernel and Language for Software Engineering Methods" [30, 31]. Стандарт вводит понятия ядра (базовые понятия, используемые в программной инженерии, такие как требования, программный комплекс и т. п.), языка (определения синтаксиса и семантики, используемых при определении новых практик инженерии программ), а также практик и составляемых из них методов. Предполагается, что с использованием этого стандарта и разрабатываемых на его основе средств поддержки появится возможность строгого описания, сравнения и внедрения существующих и вновь создаваемых методов программной инженерии в массовое производство.

В числе современных тенденций следует отметить повышенный интерес к разработке языков программирования, призванных сменить устаревшие, широко используемые в настоящее время языки программирования. В частности, языки Go (Google, [32]) и Rust (Mozilla, [33]) разрабатывают в качестве системных языков программирования, которые способны частично (в случае Go, использующего автоматическое управление памятью) или полностью (в случае Rust) заменить язык C. В рамках этой тенденции может быть выделен функциональный язык программирования Swift (Apple, [34]), который предлагается в качестве языка программирования приложений с графическим интерфейсом вместо используемого с начала 2000-х гг. языка Objective-C. Еще одним примером, иллюстрирующим отмеченную тенденцию, является появление большого числа языков программирования как предметно-ориентированных, так и общего назначения, основанных на языке и платформе JavaScript [35—38].

На основе инженерных решений, которые легли в основу платформы Java, в начале 2000-х была разработана платформа .NET. Для этой платформы была заявлена возможность статического анализа промежуточного кода в целях определения безопасности доступа к памяти в программе [39]. Кроме того, следует отметить появление и распространение средств статического анализа для традиционных языков, в особенности языка C [40—43].

Вероятно, важнейшим результатом практического применения средств формальной верификации является разработка верифицированного микроядра операционной системы SeL4 [44]. В работе [45] рассмотрена подробная ретроспектива проекта (в том числе с обзором предшествовавших проектов по верификации ядер ОС), которая позволяет оценить трудозатраты на верификацию относительно объема кода. Формальная спецификация системы безопасности ядра занимает 3 тыс. строк на специализированном предметно-ориентированном языке. Реализация микроядра занимает около 10 тыс. строк кода на языке C. Общий объем доказательств, гарантирующих выполнение микроядром спецификаций системы безопасности, составил 100 тыс. строк. Следует отметить, что для верификации ядра SeL4 использовалось средство Isabelle/HOL, основанное на логике высшего порядка, что показывает возможность применения для аналогичных задач средств, основанных на λ-исчислении с зависимыми типами.

В числе важных результатов по приложению λ-исчисления с зависимыми типами к задаче формальной верификации программного обеспечения в первую очередь следует отметить проект CompCert [46] — сертифицирующий компилятор языка C, разработанный с использованием средства автоматизации и поддержки математических доказательств Coq. В рамках этого проекта выпущено большое число публикаций по верификации различных аспектов процесса трансляции языка программирования (например, работы [47—49]; полный список публикаций на официальном сайте [50]). Заметим, что при реализации CompCert использовался, в том числе, адаптированный к исчислению конструкций подход к описанию семантики на основе монад, который упоминался ранее. В числе близких результатов других исследований можно выделить компилятор языка C, позволяющий верифицировать требования к вычислительной сложности используемых алгоритмов (CerCo [51]), а также исследовательский язык F^* [52], на котором была построена верифицированная реализация распределенного протокола авторизации. С учетом широкого использования методов формальной верификации при разработке аппаратного обеспечения, результатов исследований по построению формальной модели семантики набора команд и схемы работы с памятью распространенных процессорных архитектур [53, 54], а также подходов к верификации методов оптимизации программ [55, 56] можно утверждать, что принципиально возможна разработка верифицирующего компилятора языков программирования в машинный код, при использовании которого гарантируется сохранение доказанных свойств программы при исполнении на целевой аппаратной платформе [57].

Область приложения средств формальной верификации не ограничена исключительно разработкой ПО с высокими требованиями к безопасности. Так, в работе [58] представлено доказательство теоремы о четырех красках в системе Coq, которое в то же время является программой, реализующей требуемую раскраску удовлетворяющих условиям теоремы графов. Этот пример является практической иллюстрацией соответствия Карри—Говарда, упомянутого во введении. На основе средства Coq были реализованы также элементы теории доменов [59].

Выделим несколько направлений теоретических исследований по расширению исчисления конструкций. Расширения моделей индуктивных типов вложенными индуктивными [60], индексированными индуктивными [61], индуктивно-рекурсивными [62] и индуктивно-индуктивными [63] типами снимают часть ограничений исчисления, связанных с описанием корректных по построению сложных по своей структуре программ. Кроме того, результаты последних исследований по семантике таких расширений [64, 65] позволяют упростить реализацию индуктивных и коиндуктивных типов. Одним из современных направлений исследований является гомотопическая теория типов [66] и связанная с ней программа унивалентных оснований математики [67], в рамках которой предполагается использовать расширенную теорию типов как базис для описания математических теорий. В частности, в рамках этого направления была получена модель теории типов в терминах алгебраической топологии.

Реализация средства поддержки доказательств Matita [68] на основе исчисления конструкций показывает, что возможна компактная реализация (до 5 тыс. строк кода) ядра исчисления конструкций. Такое ядро в первую очередь содержит реализацию алгоритма проверки типов, от корректности которой непосредственно зависит корректность проверки доказательств системой.

3. Российские исследования

Исследования в области формальной верификации ПО в настоящее время активно ведутся в том числе и российскими научными организациями. За исключением исследований в области методов *model checking*, которые, как было отмечено во введении, выходят за рамки настоящей статьи, в число основных подходов к описанию формальных моделей программ и формальной верификации, рассматриваемых в настоящее время российскими учеными, входят аксиоматический, операционный и денотационный подходы, а также подходы на основе схем программ. Общие вопросы построения формальных моделей программ и подходов к решению задачи их формальной верификации рассмотрены в учебном пособии [69].

Методы описания моделей программ на основе схем программ, предложенные А. П. Ершовым и Ю. И. Яновым в 1950-х гг., на настоящее время остаются в области интереса преимущественно российских ученых. Тем не менее такие методы предоставляют широкий класс возможностей по исследованию преобразований программ, в особенности при исследовании вопросов корректности различных оптимизаций при

их трансляции. В числе работ на этом направлении следует отметить работу [70] и недавние работы по разрешимости задачи эквивалентности некоторых классов схем программ [71]. Вопросы корректности преобразований программ рассмотрены также в работе [72], однако вместо подхода на основе схем программ в ней использованы логические предикаты на базе абстрактных синтаксических деревьев.

Вопросы описания статической формальной семантики языков программирования на примере стандарта ECMA-335 исследованы в работе [73]. Подходы к построению формальных моделей программ на основе денотационной семантики в приложении, в частности, к квазипараллельным и параллельным вычислениям рассмотрены в работах [74, 75]. Модели и методы, предложенные в первой работе, были использованы на практике при реализации системы динамического распараллеливания NewTS. В работе [76] для описания формальной семантики регистровых языков программирования, таких как языки ассемблера, используется операционная семантика. Модификация операционного подхода, получившая название "операционно-онтологический подход", рассмотрена в работе [77].

Известны исследования по описанию аксиоматической семантики подмножеств языков C [78—81] и C# [82]. Кроме того, в работе [83] рассмотрен пример решения задачи формальной верификации программной реализации алгоритма сортировки с использованием аксиоматической семантики в системе Isabelle/HOL. Средство автоматизированной верификации, использующее методы аксиоматической семантики и алгоритмы решения задачи выполнимости булевых формул, представлено в работе [84]. Алгебраический подход в работе [85] применяется для описания семантики императивного языка программирования на базе Pascal с нетривиальной семантикой операций присваивания. Вопросы описания аксиоматической семантики функционального потокового (*data-flow*) языка программирования были рассмотрены в работе [86].

В числе методов, развиваемых российскими научными организациями, следует упомянуть подход к автоматизированной генерации тестов на основе формальной спецификации программ UniTesK [87]. Как отмечено в работе [88], подход применялся на практике в таких областях, как тестирование операционных систем и реализаций телекоммуникационных протоколов, а также при верификации аппаратного обеспечения. Известны также результаты применения методов автоматизированной генерации тестов с использованием статической семантики языка программирования C к верификации компилятора такого языка [89].

С учетом основной тематики настоящей статьи следует отметить публикации в русскоязычных изданиях, связанные с применением языков программирования с зависимыми типами. В частности, в работе [90] на языке Agda реализованы конструктивные версии утверждений и методов коммутативной алгебры. В работе [91] представлена реализация метода решения задачи выполнимости булевых формул на языке Agda в приложении к верификации на основе темпоральной логики. Такой результат показывает возможность применения методов *model checking* в рамках верификации программ с использованием языков программирования с зависимыми типами.

Заключение

Результаты анализа исследований в области программной инженерии, реинжиниринга и верификации программного обеспечения, формальных моделей программ и методов разработки языков программирования как общего назначения, так и предметно-ориентированных, представленные в настоящей статье, а также в статье "Формальные модели программ и языков программирования. Часть 1. Библиографический обзор 1930—1989 гг.", позволяют сделать следующие выводы.

В целом средства статического анализа в постоянном инструментарии разработчика ПО стали активно использоваться только в последние годы. Таким образом, следующее поколение средств контроля качества исходного кода ПО, которое будет внедрено в широкую практику разработки и контроля качества ПО, по всей видимости, будет основано именно на методах формальной верификации.

Тем не менее на настоящее время средства формальной верификации программ, написанных на языках программирования общего назначения, не имеют широкого распространения. Например, в рамках настоящей библиографии было перечислено несколько исследований, рассматривающих построение формальной семантики подмножества языка программирования и байт-кода Java. Однако внедрения в общую практику разработки ПО на этой платформе за 15 лет так и не произошло. При этом программы на Java используются в том числе и в областях, предъявляющих высокие требования к безопасности (смарт-карты, банковское ПО). Из этого факта может быть сделан вывод о том, что сложность языка программирования общего назначения также может являться фактором, затрудняющим формальную верификацию программ, написанных на таком языке.

Вместе с тем подход языково-ориентированного программирования требует, чтобы программные комплексы были описаны с использованием ограниченных высокоуровневых предметно-ориентированных языков программирования. В связи с этим целесообразно в качестве одного из дальнейших направлений исследований в области формальной верификации рассматривать разработку методов и средств построения предметно-ориентированных языков программирования с заданной формальной семантикой. Для таких языков возможна, в частности, реализация методов частичной автоматизации доказательств, а в некоторых случаях и сведение формальной верификации определенного класса

требований к проверке типов такого предметно-ориентированного языка. Перечисленные в разд. 2 библиографического обзора примеры новых языков программирования показывают, что разработка и ввод в эксплуатацию новых языков программирования, в том числе предметно-ориентированных, на практике возможна и используется. Создание таких языков уже выполняется крупными корпорациями, деятельность которых основана на разрабатываемом для внутреннего использования ПО, а также на создании так называемой экосистемы приложений сторонних разработчиков на базе проприетарной платформы.

Методы формальной верификации с использованием языков программирования с зависимыми типами в последние годы являются объектом интереса широкого круга исследователей. Перечисленные примеры результатов недавних исследований, полученных с использованием таких методов, в особенности компилятор CompCert и микроядро SeL4, верифицированное с использованием близкого к языкам программирования с зависимыми типами средства Isabelle/HOL, показывают возможность их практического применения для решения задач формальной верификации. Кроме того, перспективы компактной реализации алгоритмов проверки доказательств, построенных на основе языков программирования с зависимыми типами, свидетельствуют о возможности реализации фундаментального кода, несущего доказательство в терминах таких языков. С использованием такого подхода в перспективе может стать возможным, в частности, решение такой актуальной задачи, как выполнение автоматической проверки программного обеспечения на предмет отсутствия недекларированных возможностей, непосредственно при развертывании программных комплексов на целевой платформе. По этой причине основанное на λ-исчислении с зависимыми типами промежуточное представление можно рассматривать как потенциально перспективную основу для перспективных средств построения предметно-ориентированных языков с заданной формальной семантикой.

Список литературы

1. **Clarke E. M., Grumberg O., Peled D. A.** Model Checking. Cambridge, Mass.: The MIT Press, 1999. 314 p.
2. **Sørensen M. H., Urzyczyn P.** Lectures on the Curry—Howard isomorphism. V. 149 of Studies in Logic and the Foundations of Mathematics. Elsevier, 2006.
3. **Thompson S.** Type Theory and Functional Programming. Inc, Redwood City, CA. Addison Wesley Longman Publishing Co., 1991.
4. **Gamma E., Helm R., Johnson R., Vlissides J.** Design patterns: elements of reusable object-oriented software. Reading, Mass.: Addison-Wesley, 1995. 395 p.
5. **The JavaTM** Language Environment: Rep. Sun Microsystems: James Gosling, Henry McGilton. Mountain View, CA, USA, 1996. 100 p.
6. **Guthery S. B.** Java card: Internet computing on a smart card // IEEE Internet Computing. 1997. Vol. 1, N. 1. P. 57—59.
7. **Rose E., Rose K. H.** Lightweight Bytecode Verification // OOPSALA Workshop on Formal Underpinnings of Java. 1998. P. 1—23.
8. **Poetzsch-Heffter A., Muller P.** A Programming Logic for Sequential Java // European symposium on programming languages and systems (ESOP). Vol. 1576 of Lecture Notes in Computer Science. Heidelberg, Springer, 1999. P. 162—176.
9. **Owre S., Rushby J., Shankar N., von Henke F.** Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS // IEEE Transactions on Software Engineering. 1995. Vol. 21, N. 2. P. 107—125.
10. **Guaspari D., Marceau C., Polak W.** Formal verification of Ada programs // IEEE Transactions on Software Engineering. 1990. Vol. 16, N. 9. P. 1058—1075.
11. **Rushby J., von Henke F.** Formal Verification of Algorithms for Critical Systems // IEEE Transactions on Software Engineering. 1993. Vol. 19, N. 1. P. 13—23.
12. **Holzmann G. J., Peled D.** An Improvement in Formal Verification // Proceedings of the 7th IFIP WG6.1 International Conference on formal description techniques VII. 1995. P. 197—211.
13. **Meadows C. A.** Formal Verification of Cryptographic Protocols: A Survey. Advances in cryptology // ASIACRYPT'94. Berlin Heidelberg: Springer. 1995. P. 133—150.
14. **Ward M.** Language-Oriented Programming // Software — Concepts and Tools. 1994. Vol. 15, N. 4. P. 147—161.
15. **Van Deursen A., Klint P., Visser J.** Domain-Specific Languages: An Annotated Bibliography. // ACM SIGPLAN Notices. 2000. Vol. 35, N. 6. P. 26—36.
16. **Papaspyrou N. S.** A formal semantics for the c programming language. PhD thesis, National Technical University of Athenes, Athenes, Greece, 1998.
17. **Moggi E.** Notions of Computation and Monads // Information and Computation. 1991. Vol. 93, N. 1. P. 55—92.
18. **Necula G. C.** Proof-Carrying Code // Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on principles of programming languages — POPL '97. ACM Press, 1997. P. 106—119.
19. **Appel A.** Foundational Proof-Carrying Code // Proceedings 16th annual IEEE symposium on logic in computer science. IEEE Comput. Soc, 2001. P. 247—256.
20. **Pfenning F., Paulin-Mohring C.** Inductively Defined Types in the Calculus of Constructions. // Proceedings of mathematical foundations of programming semantics. Springer-Verlag, 1990. P. 209—228.
21. **Paulin-Mohring C.** Inductive Definitions in the System Coq: Rules and Properties // TLCA '93. Proceedings of the International Conference on typed Lambda Calculi and Applications. Springer-Verlag London, 1993. P. 328—345.
22. **Altenkirch T.** Constructions, inductive types and strong normalization. PhD thesis, University of Edinburgh, Edinburhg, UK. 1993.
23. **Mendler N. P.** Inductive Types and Type Constraints in the Second-Order Lambda Calculus. // Annals of Pure and Applied Logic. 1991. Vol. 51, N. 1—2. P. 159—172.
24. **Gimenez E.** Codifying Guarded Definitions with Recursive Schemes // Types for proofs and programs: International workshop TYPES'94. Bastad, Sweden, June 6—10, 1994. Selected Papers. Springer, 1995. P. 39—59.
25. **Huet G.** The Gilbreath Trick: A Case Study in Axiomatisation and Proof Development in the Coq Proof Assistant // Proceedings of the second workshop on logical frameworks. Rapport de recherche INRIA 1511, September. 1991. URL: http://yquem.inria.fr/~huet/PUBLIC/shuffle2.pdf
26. **Критически** важные объекты и кибертерроризм. Часть 1. Системный подход к организации противодействия / Под ред. В. А. Васенина. М.: МЦНМО, 2008. 398 с.
27. **Критически** важные объекты и кибертерроризм. Часть 2. Аспекты программной реализации средств противодействия / Под ред. В. А. Васенина. М.: МЦНМО, 2008. 607 с.
28. **Jacobson I., Bertr M., Soley R.** The SEMAT Initiative: A Call for Action. URL: http://www.drdobbs.com/architecture-and-design/the-semat-initiative-a-call-for-action/222001342 (дата обращения 24.02.2015).
29. **SEMAT.** Software Engineering Method and Theory. 2015. URL: http://semat.org/ (дата обращения 24.02.2015).
30. **Jacobson I., Pan-Wei Ng, McMahon P.** et al. The Essence of Software Engineering: The SEMAT Kernel // Communications of the ACM. 2012. Vol. 55, N. 12. P. 42—49.
31. **OMG.** Kernel and Language for Software Engineering Methods (Essence). OMG Document N. formal/2014-11-02. Version 1.0 edition. Object Management Group, 2014. November.

32. **Pike R**. Go at Google // Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity. SPLASH '12. New York, NY, USA: ACM, 2012. P. 5—6.
33. **The Rust** Programming Language. URL: http://www.rust-lang.org (дата обращения 24.02.2015).
34. **Deitel P. J., Deitel H. M., Deitel A.** iOS 8 for Programmers: An App-Driven Approach with Swift. 3rd edition. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. 450 p.
35. **Muscar A.** Agents in the Browser: Using Agent Oriented Programming for Client Side Development // Recent Developments in Computational Collective Intelligence. V. 513 of Studies in Computational Intelligence. Springer, 2014. P. 79—90.
36. **Wilkinson S. R., Almeida J. S.** QMachine: commodity supercomputing in web browsers // BMC bioinformatics. 2014. Vol. 15, N. 1. P. 176.
37. **McKenna B.** Roy: A Statically Typed, Functional Language for JavaScript // IEEE Internet Computing. 2012. N. 3. P. 86—91.
38. **Nicolae C. E.** List of Languages that Compile to JS, URL: https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS (дата обращения 24.02.2015).
39. **Richter J.** Applied Microsoft. NET Framework Programming. Redmond: Microsoft Press Redmond, 2002. 640 p.
40. **Software** Testing and Static Analysis Tools. URL: https://www.coverity.com/ (дата обращения 24.02.2015).
41. **Статический** анализатор кода для C и C++. URL: http://www.viva64.com/ru (дата обращения: 24.02.2015).
42. **Cuoq P., Kirchner F., Kosmatov N. et al.** Frama-C // Software Engineering and Formal Methods. V. 7504 of Lecture Notes in Computer Science. Springer, 2012. P. 233—247.
43. **Kremenek T.** Finding software bugs with the Clang Static Analyzer. California: Apple Inc. URL: http://llvm.org/devmtg/2008-08/Kremenek_StaticAnalyzer.pdf (дата обращения: 24.02.2015).
44. **Klein G., Elphinstone K., Heiser G.** et al. seL4: Formal Verification of an OS Kernel // Proceedings of the ACM SIGOPS 22Nd symposium on operating systems principles. New York, NY, USA: ACM, 2009. P. 207—220.
45. **Klein G., Andronick J., Elphinstone K.** et al. Comprehensive Formal Verification of an OS Microkernel // ACM Trans. Comput. Syst. 2014. Vol. 32, N. 1. P. 2:1—2:70.
46. **Leroy X.** Formal Verification of a Realistic Compiler // Communications of the ACM. 2009. Vol. 52, N. 7. P. 107—115.
47. **Maroneze A.** Verified compilation and worst-case execution time: PhD thesis. Université de Rennes 1. 2014.
48. **Blazy R., Dargaye Z., Leroy X**. Formal Verification of a C Compiler Front-End // FM 2006: Formal Methods. Lecture Notes in Computer Science 4085. Springer Berlin Heidelberg, 2006. P. 460—475.
49. **Jourdan J.-H., Leroy X., Pottier F.** Validating LR(1) Parsers // Proceedings of the 21st European Symposium on Programming. 2012. Vol. 7211. P. 397—416.
50. **CompCert —** Publications. URL: http://compcert.inria.fr/publi.html\#chapters (дата обращения: 25.02.2015).
51. **Amadio R. M., Ayache N., Bobot F.** et al. Certified Complexity (CerCo) // Foundational and practical aspects of resource analysis. Lecture Notes in Computer Science / Eds. by U. D. Lago, R. Pena. Springer International Publishing, 2013. P. 1—18.
52. **Strub P.-Y., Swamy N., Fournet C., Chen J.** Self-Certification: Bootstrapping Certified Typecheckers in F* with Coq // ACM SIGPLAN notices. 2012. Vol. 47. P. 571—584.
53. **Alglave J., Fox A., Ishtiaq S. et al.** The Semantics of Power and ARM Multiprocessor Machine Code // Proceedings of the 4th workshop on declarative aspects of multicore programming. ACM, 2009. P. 13—24.
54. **Sewell P., Sarkar S., Owens S. et al.** X86-TSO: A Rigorous and Usable Programmer's Model for X86 Multiprocessors // Communications of the ACM. 2010. Vol. 53, N. 7. P. 89—97.
55. **Bansal S., Aiken A.** Automatic generation of peephole superoptimizers // ACM SIGPLAN notices. 2006. Vol. 41. P. 394—403.
56. **Tate R.** Equality saturation: Using equational reasoning to optimize imperative functions. PhD thesis, University of California, San Diego, USA. 2012.
57. **Appel A. W.** Verified software toolchain // Programming Languages and Systems. Springer, 2011. P. 1—17.
58. **Gonthier G.** Formal proof — the four-color theorem // Notices of the AMS. 2008. Vol. 55, N. 11. P. 1382—1393.
59. **Benton N., Kennedy A., Varming C.** Some Domain Theory and Denotational Semantics in Coq // Theorem proving in higher order logics. Lecture Notes in Computer Science N. 5674 / Eds. by S. Berghofer, T. Nipkow, C. Urban, M. Wenzel. Springer Berlin Heidelberg, 2009. P. 115—130.
60. **Matthes R.** An induction principle for nested datatypes in intensional type theory // J. Funct. Program. 2009. Vol. 19. P. 439—468.
61. **Benke M., Dybjer P., Jansson P.** Universes for generic programs and proofs in dependent type theory // Nord. J. Comput. 2003. Vol. 10, N. 4. P. 265—289.
62. **Ghani N., Hancock P**. An Algebraic Foundation and Implementation of Induction Recursion and Indexed Induction Recursion. URL: https://personal.cis.strath.ac.uk/neil.ghani/papers/ghani-mscs11.pdf (дата обращения 25.02.2015).
63. **Forsberg F. N., Setzer A.** Inductive-inductive definitions // Computer Science Logic. Springer, 2010. P. 454—468.
64. **Fumex C., Ghani N., Johann P.** Indexed induction and coinduction, fibrationally // Algebra and Coalgebra in Computer Science. Springer, 2011. P. 176—191.
65. **Ghani N., Malatesta L., Forsberg F.N., Setzer A.** Fibred Data Types // Proceedings of the 2013 28th annual ACM/IEEE symposium on logic in computer science. IEEE Computer Society, 2013. P. 243—252.
66. **The Univalent** Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics. Princeton, NJ: Institute for Advanced Study, 2013.
67. **Voevodsky V.** Univalent foundations project. NSF grant application. URL: http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/univalent_foundations_project.pdf (дата обращения 25.02.2015).
68. **Asperti A., Ricciotti W., Coen C. S., Tassi E.** A compact kernel for the calculus of inductive constructions // Sadhana. 2009. Vol. 34, N. 1. P. 71—144.
69. **Шилов Н. В.** Основы синтаксиса, семантики, трансляции и верификации программ: учеб. пособие. Новосибирск: Изд-во НГУ, 2011. 292 с.
70. **Подловченко Р. И., Горская И. В., Захаров В. А.** и др. Исследование проблем эквивалентности и эквивалентных преобразований программ методами теории схем программ и неклассических логик: отчет о НИР/НИОКР: 94-01-00054-а. МГУ ВМиК, 1994.
71. **Подловченко Р. И., Молчанов А. Э.** Разрешимость эквивалентности в перегородчатых моделях программ // Моделирование и анализ информационных систем. 2014. Т. 21, № 2. С. 56—70.
72. **Нис З. Я.** Преобразования программ: контроль семантической корректности // Известия высших учебных заведений. Северо-Кавказский регион. Серия: Естественные науки. 2010. № 1. С. 18—21.
73. **Васенин В. А., Кривчиков М. А.** Статическая семантика стандарта ECMA-335 // Программирование. 2012. № 4. С. 3—16.
74. **Водомеров А. Н.** Построение формальной модели Т-системы и исследование ее корректности // Вычислительные методы и программирование: новые вычислительные технологии. 2006. Т. 7, № 2. С. 71—78.
75. **Васенин В. А., Кривчиков М. А.** Модель динамического параллельного исполнения программ // Программирование. 2013. № 1. С. 45—59.
76. **Кудрявцева И. А.** Формальная операционная семантика модельных регистровых языков программирования // Новые образовательные стратегии в современном информационном пространстве. СПб.: Изд-во РГПУ, 2007. С. 170—176.
77. **Ануреев И. С.** Операционно-онтологический подход к формальной спецификации языков программирования // Программирование. 2009. Т. 35, № 1. С. 50—60.
78. **Непомнящий В. А., Ануреев И. С., Промский А. В.** На пути к верификации C программ. Аксиоматическая семантика C-kernel // Программирование. 2003. Т. 29, № 6. С. 65—80.
79. **Ануреев И. С., Марьясов И. В., Непомнящий В. А.** Верификация C-программ на основе смешанной аксиоматической семантики // Моделирование и анализ информационных систем. 2010. Т. 17, № 3. С. 5—28.
80. **Promsky A. V.** A formal approach to the error localization Preprint: 169 / IIS SB RAS: Novosibirsk: 2012. 32 p.
81. **Кондратьев Д. А., Промский А. В.** Разработка самоприменимой системы верификации. Теория и практика //

Моделирование и анализ информационных систем. 2014. Т. 21, № 6. С. 71—82.

82. **Непомнящий В. А., Ануреев И. С., Дубрановский И. В., Промский А. В.** На пути к верификации C#-программ: трехуровневый подход // Программирование. 2006. Т. 32, № 4. С. 4—20.

83. **Ковалев М. С., Далингер Я. М., Мяготин А. В.** Формальная верификация программной реализации алгоритма пирамидальной сортировки на языке СИ-0. // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. Информатика. Телекоммуникации. Управление. 2010. Т. 103. № 4. С. 83—92.

84. **Шилов Н. В.** Пример верификации в проекте F@BOOL@, основанном на булевских решателях // Моделирование и анализ информационных систем. 2010. Т. 17, № 4. С. 111—124.

85. **Замулин А. В.** Алгебраическая семантика императивного языка программирования // Программирование. 2003. Т. 29, № 6. С. 51—64.

86. **Кропачева М. С., Легалов А. И.** Формальная верификация программ, написанных на функционально-потоковом языке параллельного программирования // Моделирование и анализ информационных систем. 2012. Т. 19, № 5. С. 81—99.

87. **Кулямин В. В., Петренко А. К., Косачев А. С., Бурдонов И. Б.** Подход UniTesK к разработке тестов // Программирование. 2003. Т. 29, № 6. С. 25—43.

88. **Кулямин В. В., Петренко А. К.** Развитие подхода к разработке тестов UniTESK // Труды Института системного программирования РАН. 2014. Т. 26, № 1. С. 9—26.

89. **Архипова М. В.** Генерация тестов для семантических анализаторов // Вычислительные методы и программирование: новые вычислительные технологии. 2006. Т. 7, № 2. С. 55—70.

90. **Мешвелиани С. Д.** О зависимых типах и интуиционизме в программировании математики // Программные системы: теория и приложения. 2014. Т. 5, № 3. С. 27—50.

91. **Малаховски Я. М., Корнеев Г. А.** Применение зависимых систем типов со структурной индукцией для верификации реактивных программ // Научно-технический вестник информационных технологий, механики и оптики. 2012. № 6. С. 63—67.

V. A. Vasenin, Professor, e-mail: vasenin@msu.ru, **M. A. Krivchikov**, Researcher, e-mail: maxim.krivchikov@gmail.com, Institute of Mechanics, Lomonosov Moscow State University, Moscow

Formal Models of Programming Languages and Programs. Part 2. Present State of Research

As noted in the first part of the present article, methods of the formal verification are in the focus of scientific research since the 1960s. Such methods, however, still have to become an accepted part of software engineering verification toolbox. Approaches to the formal verification are usually divided into two distinct classes: model-theoretic and proof-theoretic. Model-theoretic approaches, most commonly referred as the model checking, are concerned with the extensive automatic check of some finitely describable model of the program states. Proof-theoretic, or deductive verification approaches, are focused on constructing proofs of target program desirable properties as derivations in a certain formal system. One of deductive verification branches having a lot of recent research activity is the dependently-typed programming. Based on informal observation of the intuitionistic natural deduction interpretability in terms of typed lambda-calculus, it is possible to create a proof-automation system using an expressive statically-typed functional programming language as its specification language. Formal semantics of both the general purpose and domain-specific programming languages can be described in terms of interpreters based on such a system. This representation may be utilized to simplify the process of the formal model elaboration for the software of interest.

The main contribution of the present article consists of literature review and critical analysis of the present state of research in the area of the formal verification, formal models of software and formally verifiable software development, based on the research results published in the time frame since 1990 to the present time. Publications on the listed topics are presented in a context of the software engineering development in general. The review is mainly concerned with the dependently-typed programming development and its application to the formal verification. Fundamental results associated with the main topic are also mentioned. In the conclusion possible directions of the further research is proposed. One of the mentioned further research areas of interest is the application of the formal verification to the language-oriented software engineering paradigm in form of domain-specific languages which have formal semantics defined by means of the generic dependently-typed intermediate representation.

Keywords: *formal verication, programming languages, domain-specific languages, programming language formal semantics, software engineering, bibliography*

References

1. **Clarke E. M., Grumberg O., Peled D. A.** *Model Checking.* Cambridge, Mass: The MIT Press, 1999. 314 p.

2. **Sorensen M. H., Urzyczyn P.** *Lectures on the Curry—Howard isomorphism.* V. 149 of Studies in Logic and the Foundations of Mathematics. Elsevier, 2006.

3. **Thompson S.** *Type Theory and Functional Programming.* Addison Wesley Longman Publishing Co., Inc, Redwood City, CA, 1991.

4. **Gamma E., Helm R., Johnson R., Vlissides J.** *Design patterns: elements of reusable object-oriented software.* Reading, Mass.: Addison-Wesley, 1995. 395 p.

5. **The JavaTM** Language Environment: Rep. Sun Microsystems: James Gosling, Henry McGilton. Mountain View, CA, USA: 1996. 100. p.

6. **Guthery S. B.** Java card: Internet computing on a smart card. *IEEE Internet Computing*, 1997, vol. 1, no. 1, pp. 57—59.

7. **Rose E., Rose K. H.** Lightweight Bytecode Verification. *OOPSALA Workshop on Formal Underpinnings of Java*, 1998, pp. 1—23.

8. **Poetzsch-Heffter A., Muller P.** A Programming Logic for Sequential Java. *European symposium on programming languages and systems (ESOP), vol. 1576 of Lecture Notes in Computer Science, Heidelberg, Springer.* 1999, pp. 162—176.

9. **Owre S., Rushby J., Shankar N., von Henke F.** Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. *IEEE Transactions on Software Engineering*, 1995, vol. 21, no. 2, pp. 107—125.

10. **Guaspari D., Marceau C., Polak W.** Formal Verification of Ada Programs. *IEEE Transactions on Software Engineering*, 1990, vol. 16, no. 9, pp. 1058—1075.

11. **Rushby J., von Henke F.** Formal Verification of Algorithms for Critical Systems. *IEEE Transactions on Software Engineering*, 1993, vol. 19, no. 1, pp. 13—23.

12. **Holzmann G. J., Peled D.** An Improvement in Formal Verification. *Proceedings of the 7th IFIP WG6.1 International Conference on formal description techniques VII*, 1995, pp. 197—211.

13. **Meadows C. A.** Formal Verification of Cryptographic Protocols: A Survey. Advances in cryptology. *ASIACRYPT'94*. Springer Berlin Heidelberg, 1995, pp. 133—150.

14. **Ward M.** Language-Oriented Programming. *Software — Concepts and Tools*, 1994, vol. 15, no. 4, pp. 147—161.

15. **Van Deursen A., Klint P., Visser J.** Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices,* 2000, vol. 35, no. 6, pp. 26—36.

16. **Papaspyrou N. S.** A formal semantics for the c programming language. PhD thesis, National Technical University of Athenes, Athenes, Greece, 1998.

17. **Moggi E.** Notions of Computation and Monads. *Information and Computation*, 1991, vol. 93, no. 1, pp. 55—92.

18. **Necula G. C.** Proof-Carrying Code. *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on principles of programming languages — POPL '97*. ACM Press, 1997, pp. 106—119.

19. **Appel A.** Foundational Proof-Carrying Code. *Proceedings 16th annual IEEE symposium on logic in computer science*. IEEE Comput. Soc, 2001, pp. 247—256.

20. **Pfenning F., Paulin-Mohring C.** Inductively Defined Types in the Calculus of Constructions. *Proceedings of mathematical foundations of programmi NG semantics*, Springer-Verlag, 1990, pp. 209—228.

21. **Paulin-Mohring C.** Inductive Definitions in the System Coq: Rules and Properties. *TLCA '93 proceedings of the international conference on typed lambda calculi and applications,* Springer-Verlag London, 1993, pp. 328—345.

22. **Altenkirch T.** Constructions, inductive types and strong normalization. PhD thesis, University of Edinburgh, Edinburhg, UK, 1993.

23. **Mendler N. P.** Inductive Types and Type Constraints in the Second-Order Lambda Calculus. *Annals of Pure and Applied Logic,* 1991, vol. 51, no. 1—2, pp. 159—172.

24. **Gimenez E.** Codifying Guarded Definitions with Recursive Schemes. *Types for proofs and programs: International workshop TYPES'94*, Bastad, Sweden, June 6—10, 1994. Selected Papers. Springer, 1995, pp. 39—59.

25. **Huet G.** The Gilbreath Trick: A Case Study in Axiomatisation and Proof Development in the Coq Proof Assistant. *Proceedings of the second workshop on logical frameworks*, Rapport de recherche INRIA 1511, September, 1991, available at: http://yquem.inria.fr/~huet/PUBLIC/shuffle2.pdf

26. **Kriticheski** *Vazhnye ob'ekty I kiberterrorizm. Chast' 1. Sistemnyy podkhod k organizatsii Protivodeystviya* (Critically Important Objects and Cyberterrorism. Part I: System Approach to Counteraction) / Eds. by V. A. Vasenin. Moscow: MCCME Publishing, 2008, 398 p. (in Russian).

27. **Kriticheski** *vazhnye ob'ekty I kiberterrorizm. Chast' 2. Aspekty programmnoy realizatsii sredstv protivodeystviya* (Critically important objects and cyberterrorism. Part II: Implementation aspects of counteraction's software tools) / Eds. by V. A. Vasenin Moscow: MCCME Publishing, 2008, 607 p. (in Russian).

28. **Jacobson I., Meyer B., Soley R.** The SEMAT Initiative: A Call for Action, available at: http://www.drdobbs.com/architecture-and-design/the-semat-initiative-a-call-for-action/222001342.

29. **SEMAT.** Software Engineering Method and Theory, available at: http://semat.org/

30. **Jacobson I., Pan-Wei Ng, McMahon P., Spence I., Lidman S.** The Essence of Software Engineering: The SEMAT Kernel. *Communications of the ACM*, 2012, vol. 55, no. 12, pp. 42—49.

31. **OMG.** Kernel and Language for Software Engineering Methods (Essence). Object Management Group, 2014.

32. **Pike R.** Go at Google. *Proceedings of the 3rd annual conference on systems, programming, and applications: Software for humanity*. New York, NY, USA: ACM, 2012, pp. 5—6.

33. **The** Rust Programming Language, available at: http://www.rust-lang.org/

34. **Deitel P. J., Deitel H. M., Deitel A.** *iOS 8 for Programmers: An App-Driven Approach with Swift*. 3rd edition. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014.

35. **Muscar A.** Agents in the Browser: Using Agent Oriented Programming for Client Side Development. *Recent developments in computational collective intelligence*, Springer, 2014, pp. 79—90.

36. **Wilkinson S. R., Almeida J. S.** QMachine: Commodity Supercomputing in Web Browsers. *BMC bioinformatics*, 2014, vol. 15, no. 1, pp. 176.

37. **McKenna B.** Roy: A Statically Typed, Functional Language for JavaScript. *IEEE Internet Computing,* 2012, no. 3, pp. 86—91.

38. **Nicolae C. E.** List of Languages that Compile to JS, available at: https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS

39. **Richter J.** *Applied Microsoft. NET Framework Programming. Redmond*: Microsoft Press Redmond, 2002. 640 p.

40. **Software** Testing and Static Analysis Tools, available at: https://www.coverity.com/

41. **Staticheskiy** Analizator Koda Dlya C i C++, available at: http://www.viva64.com/ru/

42. **Cuoq P., Kirchner F., Kosmatov N., Prevosto V., Signoles J., Yakobowski B.** Frama-C. *Software engineering and formal methods*. Springer, 2012, pp. 233—247.

43. **Kremenek T.** Finding Software Bugs with the Clang Static Analyzer. California: Apple Inc, available at: http://llvm.org/devmtg/2008-08/Kremenek_StaticAnalyzer.pdf.

44. **Klein G., Elphinstone K., Heiser G., Andronick J., Cock D., Derrin P., Elkaduwe D., Engelhardt K., Kolanski R., Norrish M., Sewell T., Tuch H. Winwood S**. seL4: Formal Verification of an OS Kernel. *Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles*. New York, NY, USA: ACM, 2009, pp. 207—220.

45. **Klein G., Andronick J., Elphinstone K., Murray T., Sewell T., Kolanski R. Heiser G.** Comprehensive Formal Verification of an OS Microkernel. *ACM Trans. Comput. Syst.,* 2014, vol. 32, no. 1, pp. 2:1—2:70.

46. **Leroy X.** Formal Verification of a Realistic Compiler. *Communications of the ACM*, 2009, vol. 52, no. 7, pp. 107—115.

47. **Maroneze A.** Verified compilation and worst-case execution time : PhD thesis. Université de Rennes 1. 2014. June.

48. **Blazy R., Dargaye Z., Leroy X.** Formal Verification of a C Compiler Front-End. *FM 2006: Formal Methods. Lecture Notes in Computer Science 4085*. Springer Berlin Heidelberg, 2006, pp. 460—475.

49. **Jourdan J.-H., Leroy X., Pottier F.** Validating LR(1) Parsers. *Proceedings of the 21st European Symposium on Programming*, 2012, vol. 7211, pp. 397—416.

50. **CompCert —** Publications, available at: http://compcert.inria.fr/publi.html\#chapters

51. **Amadio R. M., Ayache N., Bobot F., Boender J. P., Campbell B., Garnier I., Madet A., McKinna J., Mulligan D. P., Piccolo M., Pollack R., Régis-Gianas Y., Coen C. S., Stark I. Tranquilli P.** Certified Complexity (CerCo). *Foundational and practical aspects of resource analysis. Lecture Notes in Computer Science*. Eds. by U. D. Lago, R. Pena, Springer International Publishing, 2013, pp. 1—18.

52. **Strub P.-Y., Swamy N., Fournet C., Chen J.** Self-Certification: Bootstrapping Certified Typecheckers in F* with Coq. *ACM SIGPLAN notices*, 2012, vol. 47, pp. 571—584.

53. **Alglave J., Fox A., Ishtiaq S., Myreen M. O., Sarkar S., Sewell P., Nardelli F. Z.** The Semantics of Power and ARM Multiprocessor Machine Code. *Proceedings of the 4th workshop on declarative aspects of multicore programming,* ACM, 2009, pp. 13—24.

54. **Sewell P., Sarkar S., Owens S., Nardelli F. Z., Myreen M. O.** X86-TSO: A rigorous and usable Programmer's model for X86

multiprocessors. *Communications of the ACM*, 2010, vol. 53, no. 7, pp. 89—97.

55. **Bansal S., Aiken A.** Automatic Generation of Peephole Superoptimizers. *ACM SIGPLAN notices*, 2006, vol. 41, pp. 394—403.

56. **Tate R.** Equality saturation: Using equational reasoning to optimize imperative functions. PhD thesis, University of California, San Diego, USA. 2012.

57. **Appel A. W.** Verified Software Toolchain. *Programming languages and systems*. Springer, 2011, pp. 1—17.

58. **Gonthier G.** Formal Proof—the Four-Color Theorem. *Notices of the AMS*, 2008, vol. 55, no. 11, pp. 1382—1393.

59. **Benton N., Kennedy A., Varming C.** Some Domain Theory and Denotational Semantics in Coq. *Theorem proving in higher order logics. Lecture Notes in Computer Science N. 5674*. Eds. by S. Berghofer, T. Nipkow, C. Urban, M. Wenzel. Springer Berlin Heidelberg, 2009, pp. 115—130.

60. **Matthes R.** An Induction Principle for Nested Datatypes in Intensional Type Theory. *J. Funct. Program,* 2009, vol. 19, pp. 439—468.

61. **Benke M., Dybjer P., Jansson P.** Universes for Generic Programs and Proofs in Dependent Type Theory. *Nord. J. Comput.*, 2003, vol. 10, no. 4, pp. 265—289.

62. **Ghani N., Hancock P.** An Algebraic Foundation and Implementation of Induction Recursion and Indexed Induction Recursion, available at: https://personal.cis.strath.ac.uk/neil.ghani/papers/ghani-mscs11.pdf

63. **Forsberg F. N., Setzer A.** Inductive-inductive definitions. *Computer Science Logic*. Springer, 2010, pp. 454—468.

64. **Fumex C., Ghani N., Johann P.** Indexed induction and coinduction, fibrationally. *Algebra and Coalgebra in Computer Science*. Springer, 2011, pp. 176—191.

65. **Ghani N., Malatesta L., Forsberg F. N., Setzer A.** Fibred Data Types. *Proceedings of the 2013 28th annual ACM/IEEE symposium on logic in computer science*. IEEE Computer Society, 2013, pp. 243—252.

66. **The Univalent** Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics. Institute for Advanced Study, Princeton, NJ, 2013.

67. **Voevodsky V.** Univalent Foundations Project. NSF Grant Application, available at: http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/univalent_foundations_project.pdf

68. **Asperti A., Ricciotti W., Coen C. S., Tassi E.** A Compact Kernel for the Calculus of Inductive Constructions. *Sadhana*, 2009. vol. 34, no. 1, pp. 71—144.

69. **Shilov N. V.** *Osnovy sintaksisa, semantiki, translyatsii i verifikatsii programm: uchebnoe posobie* (The basics of syntax, semantics, translation and verification of software: lecture notes), Novosibirsk: Izd-vo NSU, 2011. (in Russian).

70. **Podlovchenko R. I., Gozskaya I. V., Zaharov V. A.** et al. *Issledovanie problem ekvivalentnosti i ekvivalentnykh preobrazovaniy programm metodami teorii skhem programm i neklassicheskikh logik* (The research of the equivalence problems and the equivalent program transformations by means of program scheme theory and nonclassical logics). Moscow, Lomonosov Moscow State University, Research Report RFFR 94-01-00054-a, 1994 (in Russian).

71. **Podlovchenko R., Molchanov A.** Razreshimost' ekvivalentnosti v peregorodchatykh modelyakh program. *Modelirovanie i analiz informatsionnykh sistem,* 2014, vol. 21, no. 2, 2014, pp. 56—70 (in Russian).

72. **Nis Z.** Preobrazovaniya programm: kontrol' semanticheskoi korrektnosti. *Izvestiya vysshikh uchebnykh zavedeniy. Severo-Kavkazskiy region. Seriya: Estestvennye nauki*, 2010, no. 1, pp. 18—21 (in Russian).

73. **Vasenin V. A., Krivchikov M. A.** Staticheskaya semantika standarta ECMA-335 *Programmirovanie,* 2012, no. 4, pp. 3—16 (in Russian).

74. **Vodomerov A.** Postroenie formal'noi modeli T-sistemy i issledovanie ee korrektnosti. *Vychislitel'nye metody i programmirovanie: novye vychislitel'nye tekhnologii*, 2006, vol. 7, no. 2, pp. 71—78 (in Russian).

75. **Vasenin V. A., Krivchikov M. A.** Model' dinamicheskogo parallel'nogo ispolneniya programm *Programmirovanie*, 2013, no. 1, pp. 45—59 (in Russian).

76. **Kudryavtseva I.** *Novye obrazovatel'nye strategii v sovremennom informatsionnom prostranstve*. (New strategies of education in contemporary informational space), Saint Petersburg; RGPU, pp. 170—176 (in Russian).

77. **Anureev I. S.** Operatsionno-ontologicheskii podkhod k formal'noi spetsifikatsii yazykov programmirovaniya. *Programmirovanie*, 2009, vol. 35, no. 1, pp. 50—60 (in Russian).

78. **Nepomnyaschii V. A., Anureev I. S., Promskii A. V.** Na puti k verifikatsii C programm. Aksiomaticheskaya semantika C-kernel. *Programmirovanie*, 2003, vol. 29, no. 6, pp. 5—15 (in Russian).

79. **Anureev I. S., Mariasov I. V., Nepomnyaschii V. A.** Verifikatsiya C-programm na osnove smeshannoi aksiomaticheskoi semantiki. *Modelirovanie i analiz informatsionnykh sistem,* 2010, vol. 17, no. 3, pp. 5—28 (in Russian).

80. **Promskii A.** *A Formal Approach to the Error Localization*. IIS SB RAS, Novosibirsk, 2012.

81. **Kondratiev D. A., Promskii A. V.** Razrabotka samoprimenimoi sistemy verifikatsii. Teoriya i praktika. *Modelirovanie i analiz informatsionnykh sistem,* 2014, vol. 21, no. 6, pp. 71—82 (in Russian).

82. **Nepomnyaschii V. A., Anureev I. S., Dubranovskii I. V., Promskii A. V.** Na puti k verifikatsii C#-programm: trekhurovnevyi podkhod. *Programmirovanie*, 2006, vol. 32, no. 4, pp. 4—20 (in Russian).

83. **Kovalev M. S., Dalinger Ya. M., Myagotin A. V.** Formal'naya verifikatsiya programmnoi realizatsii algoritma piramidal'noi sortirovki na yazyke CI-0. *Nauchno-tekhnicheskie vedomosti Sankt-Peterburgskogo gosudarstvennogo politekhnicheskogo universiteta. Informatika. Telekommunikatsii. Upravlenie,* 2010, vol. 104, no. 103, pp. 83—92 (in Russian).

84. **Shilov N. V.** Primer verifikatsii v proekte F@BOOL@, osnovannom na bulevskikh reshatelyakh. *Modelirovanie i analiz informatsionnykh sistem,* 2010, vol. 17, no. 4, pp. 111—124 (in Russian).

85. **Zamulin A. V.** Algebraicheskaya semantika imperativnogo yazyka programmirovaniya. *Programmirovanie,* 2003, vol. 29, no. 6, pp. 51—64 (in Russian).

86. **Kropacheva M. S., Legalov A. I.** Formal'naya verifikatsiya programm, napisannykh na funktsional'no-potokovom yazyke parallel'nogo programmirovaniya. *Modelirovanie i analiz informatsionnykh sistem*, 2012, vol. 19, no. 5, pp. 81—99 (in Russian).

87. **Kuliamin V. V., Petrenko A. K., Kossatchev A. S., Burdonov I. B.** Podkhod UniTesK k razrabotke testov. *Programmirovanie,* 2003, vol. 29, no. 6, pp. 25—43 (in Russian).

88. **Kuliamin V. V., Petrenko A. K.** *Trudy Instituta sistemnogo programmirovaniya RAN,* 2014, vol. 26, no. 1, pp. 9—26 (in Russian).

89. **Arkhipova M. B.** Generatsiya testov dlya semanticheskikh analizatorov. *Vychislitel'nye metody i programmirovanie: novye vychislitel'nye tekhnologii*, 2006, vol. 7, no. 2, pp. 55—70 (in Russian).

90. **Meshveliani S. D.** O zavisimykh tipakh i intuitsionizme v programmirovanii matematiki. *Programmnye sistemy: teoriya i prilozheniya,* 2014, vol. 5, no. 3, pp. 27—50 (in Russian).

91. **Malakhovski Ya. M., Korneev G. A.** Primenenie zavisimykh sistem tipov so strukturnoi induktsiei dlya verifikatsii reaktivnykh programm. *Nauchno-tekhnicheskiy vestnik informatsionnykh tekhnologiy, mekhaniki i optiki,* 2012, no. 6, pp. 63—67 (in Russian).

УДК 004.4

Ю. А. Киселёв, мл. науч. сотр., e-mail: ykiselev.loky@gmail.com,
С. В. Поршнев, д-р техн. наук, проф., зав. кафедрой, e-mail: sergey_porshnev@mail.ru,
М. Ю. Мухин, д-р филол. наук, доц., проф., e-mail: mfly@sky.ru, Уральский федеральный университет имени первого Президента России Б. Н. Ельцина, г. Екатеринбург

Современное состояние электронных тезаурусов русского языка: качество, полнота и доступность

В настоящей обзорно-аналитической статье рассмотрены проблемные вопросы и характерные особенности электронных тезаурусов русского языка. Проанализированы качество, полнота, возможность свободного использования электронных тезаурусов. Выводы о качестве тезаурусов сделаны на основе анализа словников, а также полноты представленных синонимических отношений. Анализ выявил необходимость в разработке методов, позволяющих частично автоматизировать создание электронного тезауруса русского языка.

Ключевые слова: *тезаурус, ворднет, лексический ресурс, русский язык*

Введение

Под *тезаурусом* будем понимать особый вид словаря, отражающего семантические отношения между словами. В разных случаях тезаурусы называют идеографическими или тематическими словарями. В настоящее время все более популярными становятся *электронные тезаурусы* (ЭТ), которые уже много лет эффективно применяют в различных задачах автоматической обработки текстов. Их используют для снятия лексической многозначности, для расширения перечня слов при формулировке поисковых запросов, для автоматической классификации/рубрикации информации и др. [1]. Отметим, что для английского языка создан *WordNet* — ЭТ, который существует и поддерживается на протяжении нескольких десятилетий и при этом является бесплатным для использования [2]. Выбранная стратегия его развития оказалась столь удачной, что де-факто этот ресурс уже многие годы является стандартом, определяющим требования к ЭТ, разрабатываемым на материале других языков.

Необходимо отметить, что при анализе качества ЭТ возникают многочисленные сложности. Они связаны с выбором достаточного числа количественных характеристик, с разработкой методов их получения и последующим оцениванием. Например, при оценке качества отношений, представленных в ЭТ, требуется предложить метод, на основе которого появляется возможность сравнить ресурсы между собой. Для этой цели, на взгляд авторов, наилучшим образом подходят метод экспертных оценок [3] и метод анализа иерархий Саати [4]. Однако для их применения необходимо обосновать критерии для отбора экспертов и затем найти тех из их числа, которые отвечают данным критериям, что представляет собой далеко не простую задачу. Из представленных выше соображений следует, что задача полномасштабного многомерного анализа ЭТ по своей сложности не уступает задаче создания самого ЭТ. Однако сравнительный анализ ЭТ востребован уже сейчас. В связи с этим представляется целесообразным выбрать некоторый конечный набор количественных характеристик ЭТ, позволяющих делать выводы о качестве рассматриваемых ресурсов.

В настоящее время развивается большое число проектов по созданию русскоязычных ЭТ. Однако какого-либо стандартного ресурса, удовлетворяющего всем требованиям пользователей данных типов ресурсов, до сих пор не создано. В связи с этим оценка современного состояния ЭТ русского языка, являющаяся одной из целей исследования, результаты которого представлены в статье, актуальна. В настоящей работе представлен сравнительный анализ русскоязычных ЭТ с точки зрения возможности их использования в прикладных задачах.

1. Краткая история развития электронных тезаурусов

Идеографические словари и тезаурусы создают для разных языков вот уже почти несколько веков. Начало истории их создания датируется 1852 г., когда П. М. Роже выпустил идеографический словарь, впервые употребив термин "тезаурус" применительно к словарям. Идеи, заложенные П. М. Роже, были развиты и усовершенствованы зарубежными лексикографами, такими как П. Буассьер, Ф. Дорнзайф, Х. Касарес и др.

Русские идеографические словари начали появляться относительно недавно. В 1980-е гг. этому немало способствовали работы российских ученых Ю. Н. Караулова и В. В. Морковкина (см., например, работы [5, 6]). Так, в настоящее время для русского языка предложены различные схемы категоризации русской лексики: Идеографический словарь О. С. Баранова [7], Русский семантический словарь под ред. Н. Ю. Шведовой [8] и др. Указанные ресурсы, как кажется на первый взгляд, позволяют эффективно решать различные задачи автоматической обработки текста, например, снятие лексической многозначности слов [1]. Однако, как показали попытки их использования при создании ЭТ, все они изначально были предназначены для применения человеком, но не ЭВМ [9]. По этой причине в автоматической обработке текстов они оказались недостаточно удобными, вследствие чего примеры их использования на практике достаточно редки. В связи с этим были продолжены поиски подходов, призванных устранить отмеченные недостатки.

На рубеже 1980—1990-х гг. был создан лексико-семантический ресурс нового типа, в основу которого положена база данных концептуальных отношений, созданная на основе модели ментального лексикона человека[1]. Разработанный Джорджем Миллером (*George Miller*) и его коллегами ресурс получил название WordNet [10]. Данный ресурс в настоящее время весьма популярен. Он один из наиболее авторитетных де-факто и является стандартом в области построения лексико-семантических баз данных.

В настоящее время *ворднетами* называют любые лексические ресурсы, которые сходны по своей структуре с Принстонским WordNet (далее PWN — Princeton WordNet) [9]. Несмотря на то что PWN создавался для английского языка, на настоящее время существует большое число аналогичных ресурсов для других языков. Русский язык здесь не является исключением. Результаты сравнительного анализа особенностей наиболее известных ЭТ русского языка приведены в следующем разделе.

2. Сравнительный анализ электронных тезаурусов русского языка

Для проведения сравнительного анализа ЭТ русского языка были выбраны: переводные ворднеты и оригинальные российские разработки; ЭТ, которые больше не поддерживаются; ЭТ, которые продолжают развиваться. Такой выбор, с точки зрения авторов, позволил получить достаточно полную картину данной предметной области. В качестве показателей, по которым проводилось сравнение, были использованы цели создания ЭТ, его ключевые особенности, отличия от прототипа (PWN), количественные показатели ЭТ.

[1] Эта работа поводилась в лаборатории когнитивистики Принстонского университета, США.

Проект **RussNet** [11] — разработка исследовательской группы кафедры математической лингвистики Санкт-Петербургского государственного университета, начатая в 1999 г. Главные цели данного проекта [12]:

- отражение организации лексической системы русского языка в целом;
- представление ядерной лексики русского языка;
- выявление тех семантических, семантико-грамматических и семантико-деривационных отношений, которые являются существенными для русского языка.

В целом по своей структуре и процедуре создания проект RussNet следует подходам, принятым в PWN. При этом авторы RussNet развивают и совершенствуют принципы построения PWN. Например, национальный корпус русского языка (НКРЯ) [13] в данном проекте используется не только для выбора наиболее частотной и общеупотребительной лексики, но и для упорядочивания значений слов: значения (в терминах ворднетов — *синсеты*, т. е. наборы синонимов) перечисляются по убыванию частоты их встречаемости в корпусе. Слова в синсетах также упорядочиваются по частоте таким образом, что по синсету можно понять, какие синонимы являются более употребительными.

Авторы ресурса RussNet дорабатывают лингвистические тесты для установления типов отношений. Например, для определения синонимии используют не произвольные контексты, а реальные, взятые из НКРЯ. Авторы усиливают определение антонимии: в их подходе антонимами могут быть только те слова, которые имеют общий гипероним[2] (чтобы избежать довольно случайных противопоставлений типа: "*Собака не кусается, а играет*") [12].

Русский язык сильно отличается от английского, особенно своим многообразием словообразовательных моделей. Авторы работы [12] замечают, что доля мотивированной лексики в русском языке составляет 85 %. В RussNet учитывается деривационная синонимия (например, *чашка-чашечка*) и экспрессивная синонимия (например, *темный-темненький*). Данный ресурс представляет собой разработку русского ЭТ "с нуля"[3] авторским коллективом, в который входят лингвисты и программисты. Такой процесс подразумевает продолжительную работу в отсутствие какой-либо автоматизации труда. Проанализируем трудоемкость создания такого ресурса на примере представленной далее статистики.

В 2002 г., т. е. спустя целых три года после начала работы над проектом, было сделано следующее (все цифры примерные): всего сформировано 1000 синсетов,

[2] *Гипонимия* — это семантическое отношение между значениями слов. В отношении участвуют две сущности: *гипоним* — слово с более частным (конкретным) значением по отношению к более общему понятию, *гипериниму*.

[3] Под этим мы понимаем, что ворднет RussNet не является переводом PWN. При этом при создании данного ресурса, безусловно, используются другие лингвистические ресурсы, такие как толковые словари, тексты русского языка и пр.

охвачено 1000 существительных, 1000 глаголов, 500 прилагательных, созданы необходимые связи [14]. При этом в PWN по состоянию на 2015 г. имеется более 150 тыс. синсетов [2]. Эти цифры свидетельствуют о том, что создание ЭТ "с нуля" без применения средств автоматизации, позволяющих использовать накопленные данные аналогичных ресурсов для других языков, и осуществляемое лишь силами экспертов в области лингвистики — это очень трудозатратный процесс.

Несмотря на то что на сайте RussNet проект позиционируется как развивающийся, статистика по его наполнению соответствует той, что приводилась еще в 2003 г. [12]. К сожалению, данного ресурса нет в открытом доступе, как следствие, использование RussNet и проведение его самостоятельного анализа не представляется возможным.

Вордне-подобный электронный тезаурус русского языка РуТез [15] представляет собой еще один пример разработки ресурса, аналогичного PWN, не являющегося его переводной "копией". Проект РуТез разрабатывается в Московском государственном университете им. М. В. Ломоносова. Главной его задачей является разработка тезауруса, основным назначением которого является автоматический режим обработки текста для решения задач информационного поиска [16]. Авторский коллектив этого проекта ставит эксперименты с применением тезауруса РуТез в задачах концептуального индексирования, расширения запросов, автоматического аннотирования документов и других приложениях информационного поиска.

Данный ресурс в значительной степени отличается от PWN. Его подробное описание можно найти в монографии одного из создателей РуТез [1]. Отметим, что существенные, на наш взгляд, отличия РуТез от своего прототипа PWN состоят в следующем:

- разделение лексики на общий лексикон и общественно-политическую лексику (экономика, спорт, финансы и пр.), обусловленное, в том числе, тем, что специальная лексика в среднем гораздо более однозначна, чем общеупотребительная [16];
- использование относительно небольшого числа типов отношений (авторы выделили отношения гипонимии и меронимии[4], отношение онтологической зависимости (*кипение — жидкость*) и отношение ассоциации (*саммит — государство*).

На сайте проекта РуТез приводится статистика наполненности ресурса: 158 тыс. слов объединены в 55 тыс. понятий (значений), между которыми установлено более 210 тыс. связей. Проект РуТез до недавнего времени был закрытой разработкой, но в 2014 г. авторы выложили на сайте проекта его неполную версию в общий доступ — **РуТез-lite**. Однако отсутствует возможность скачивания ресурса. Вместо этого пользователь может вручную через браузер выборочно ознакомиться с его содержимым. Отметим, что доступные данные запрещено использовать в коммерческих целях.

Электронный тезаурус Russian WordNet (далее RWN) разрабатывался сотрудниками Петербургского университета путей сообщения и ЗАО "Руссикон" [17]. Работа над проектом шла в первой половине прошлого десятилетия и была закончена в 2005 г. Главные цели данного проекта заключались [18]:

- в построении ворднета русского языка;
- в получении методов автоматизированного построения межъязыкового индекса для англо-русской версии ворднета.

Работа над проектом RWN была разбита на два этапа. На первом этапе на основе анализа толковых словарей и словарей синонимов выделялись синсеты, которые являются базовой структурной единицей ворднетов. Авторы разработали RDFs — схему, позволяющую интегрировать разработанный ворднет с другими системами, в том числе использовать его как часть семантического веба [19].

На втором этапе устанавливалось соответствие между синсетами русского языка, выделенными на первом этапе, и синсетами PWN [18]. Для этого была использована оценочная функция, позволяющая количественно оценивать степень соответствия между синсетами русского и английского языков и выбирать из них наилучшие сочетания. Главным инструментарием, использующимся на втором этапе, стали англо-русские словари, с помощью которых осуществлялся перевод английских синсетов на русский язык.

Однако процесс создания электронного тезауруса RWN не был полностью автоматизирован. В связи с этим авторы ресурса были вынуждены использовать ручную доработку информационного ресурса [18]. Несмотря на это, по состоянию на май 2005 г., RWN охватывал более 124 тыс. слов и содержал 157 тыс. синсетов [18]. Отметим, что, будучи переводом PWN, данный ЭТ содержит существенно больше синсетов, чем оригинальный ресурс. Его создатели изначально предполагали, что RWN будет открытым информационным ресурсом. Однако этого не произошло. Данный проект оказался закрытой разработкой, и, к сожалению, установить, почему в RWN понятий больше, чем в используемой версии PWN, сейчас уже не представляется возможным.

Отметим, что в работах, связанных с решением вопросов, возникавших в ходе создания RWN, остались нерассмотренными те, которые связаны с влиянием качества используемых лингвистических ресурсов на полноту переводного ворднета. Кроме того, не был проведен анализ основных причин, препятствовавших переводу части синсетов PWN на русский язык. Как следствие, отсутствуют рекомендации по возможному улучшению качества перевода, а также методики оценки качества разработанного ресурса. В то же время данная разработка представляет определенный интерес с точки зрения возможности

[4] *Меронимия* — отношение, устанавливающее связь часть—целое между существительными. *Мероним* — это понятие, являющееся составной частью другого понятия, *холонима*.

осуществления полностью автоматического перевода ворднета на другой язык и связывания его с уже существующими ворднетами.

Другим проектом, подобным проекту RWN по использованным в нем подходам, является ЭТ, расположенный на сайте http://wordnet.ru (далее называемый просто wordnet.ru) [20]. Количественные результаты wordnet.ru также во многом похожи на результаты проекта RWN: было переведено примерно 100 тыс. синсетов. При этом в работе [20] приведена оценка качества перевода: до 75 % синсетов переведено верно. Однако оказалось, что такая точность не позволила эффективно решать задачи информационного поиска, что привело к необходимости использования дополнительной ручной разметки синсетов. Ресурсы wordnet.ru доступны для скачивания на странице проекта [21]. Отметим, что данную разработку сложно назвать ресурсом русского языка, так как, например, описания синсетов в данном ЭТ имеются только на английском, т. е. на языке оригинала. В то же время wordnet.ru оказывается единственным из рассмотренных выше ЭТ, находящимся полностью в открытом доступе.

Большой открытый ЭТ русского языка **YARN** (*Yet Another RussNet*) создается в Уральском федеральном университете совместно с Высшей школой экономики начиная с 2013 г. [22]. Главной причиной разработки проекта YARN стала (и по-прежнему остается) потребность пользователей сети Интернет в свободно распространяемом (а потому — общедоступном) и дополняемом пользователями большом ЭТ русского языка. Важное преимущество YARN в сравнении с другими ЭТ (и не только русского языка) состоит в том, что работу над ним могут осуществлять не только непосредственные участники проекта, но и заинтересованные лица с совершенно различными навыками, интересами и уровнем лингвистических знаний. Данный проект является, по существу, экспериментом, цель которого состоит в сочетании традиционных принципов создания ЭТ и вики-подхода к наполнению и редактированию лингвистических ресурсов [9]. При этом создатели проекта надеятся, что с помощью четких инструкций для наполнения, обязательного регистрирования в системе и контроля редакторов, которые будут выбираться из числа участников, им удастся добиться высокого качества разрабатываемого ЭТ [9].

В связи с тем что YARN является открытым ресурсом, одна из важных задач проекта состояла в выборе формата, используемого для распространения данных. Для ее решения авторы тезауруса разработали собственный формат представления данных, который конвертируется в формат XML и оказывается хорошо структурированным [9]. В ЭТ YARN в качестве исходных данных используется русская версия Викисловаря [23]. Из него авторы извлекли лексикон, определения и примеры к ним, а также синонимические связи. Кроме того, они используют и другие открытые лингвистические ресурсы, среди которых толковые словари, НКРЯ и др. Интерфейс, позволяющий осуществлять разметку синсетов, подробно описан в работе [24].

В то же время необходимо отметить, что данный проект пока находится на стадии становления, поэтому некоторые компоненты тезауруса (например, выделенные связи между понятиями) отсутствуют, синонимия связывает в основном имена существительные. Однако вся существующая на настоящее время разметка (более 15 тыс. синсетов) доступна для бесплатного скачивания на странице проекта [22].

Для сравнения проанализированных выше ЭТ русского языка в табл. 1 приведены их количественные показатели, информация об их доступности и возможности коммерческого использования. Здесь для ресурсов, недоступных для скачивания, использовались данные из соответствующих публикаций. При оценке количественных показателей wordnet.ru из ресурса были удалены синсеты, не содержащие русских слов, вместе с их отношениями.

Из данных табл. 1 видно, что три из шести рассмотренных ЭТ являются закрытыми разработками, свободное использование которых невозможно. Дан-

Таблица 1

Основные характеристики электронных тезаурусов русского языка

Ресурс	Число понятий, тыс.	Число отношений, тыс.	Число слов и словосочетаний, тыс.	Доступность	Разрешение коммерческого использования
RussNet [12]	5,5	8	15	Нет	Нет
Russian WordNet [18]	157	Нет данных	124	Нет	Нет
РуТез [15]	55	210	158	Нет	Нет
РуТез-lite [15]	26	108	115	Да	Нет
Wordnet.ru [21]	51,7* (117,7)	57,0 (293,3)	30,7 (130,1)	Да	Да
YARN [22]	15,4	0	27,9	Да	Да

* Здесь приводятся размеры обработанного авторами *wordnet.ru* и исходного варианта (последнее приводится в скобках)

Таблица 2

Характеристики открытых электронных тезаурусов русского языка и PWN

Характеристика	Ресурс			
	PWN	РуТез-lite	Wordnet.ru	YARN
Число слов и словосочетаний, тыс.	150,5	96,7*	31,0	27,9
Из них словосочетаний, тыс.	64,6	46,6	9,3	2,6
Число слов, имеющих заглавные буквы, тыс.	40,9	Нет данных**	0,0	0,3
Число синсетов, тыс.	98,0	25,6	25,1	16,3
Число пар слов-синонимов, тыс.	157,2	378,1	15,0	184,2

* Это значение отличается от аналогичного в табл. 1, вероятно, потому, что авторы РуТез-lite при расчете размера словника включили в него также и названия концепций из своего ресурса, которые часто описываются словосочетаниями
** Все заголовочные слова в РуТез-lite написаны заглавными буквами

ные РуТез-lite доступны для скачивания, однако их коммерческое использование запрещено лицензией. Из оставшихся двух открытых ЭТ развивающийся проект YARN в настоящее время не содержит ни одного отношения между своими понятиями, что не позволяет называть его полноценным тезаурусом. Wordnet.ru является ресурсом со смесью русского и английского языковых материалов, при этом его понятия имеют описания только на английском языке.

Таким образом, рассмотренные ЭТ по выбранным критериям существенно отличаются друг от друга, однако области применения у них одинаковые: их используют в информационном поиске, в вопросно-ответных системах, при автоматической рубрикации текстов и для решения других задач [1]. Однако разница в принципах их наполнения (вики-подход или работа внутри закрытой группы), степень открытости ресурса и другие факторы могут расширять или сужать область применения ЭТ. В этой ситуации выбор конкретного ЭТ следует осуществлять исходя из решаемых задач конкретной предметной области.

3. Анализ качества открытых электронных тезаурусов русского языка

Для более полного анализа ресурсов авторы настоящей работы рассчитали некоторые качественные характеристики ЭТ русского языка, находящихся в открытом доступе, в том числе — размер и состав словников, полноту представленных в ЭТ синонимических отношений. Для того чтобы было возможно сравнивать с наиболее известным и используемым ЭТ английского языка, были вычислены аналогичные характеристики для PWN. Данная информация представлена в табл. 2.

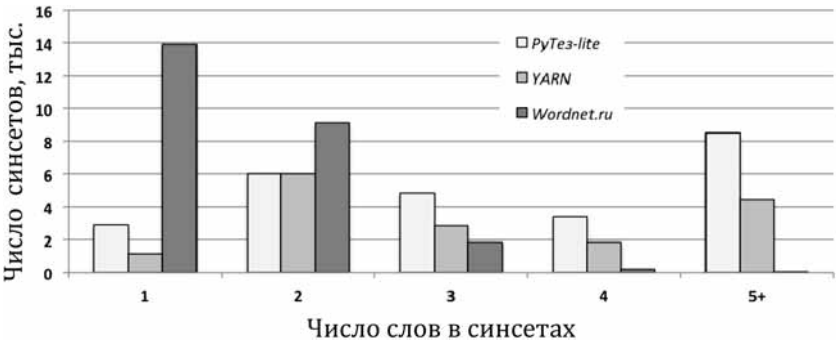
По данным, представленным в табл. 2, можно сделать следующие выводы.

1. Словник PWN существенно превосходит по размерам словники всех остальных ресурсов. При этом он содержит почти 41 тыс. слов (т. е. 27 % от их общего числа), имеющих заглавные буквы. Последнее, по мнению авторов, свидетельствует о большом количестве имен собственных в PWN, что подтверждается выборочным анализом таких слов. По этому показателю он сильно отличается от wordnet.ru, не содержащего в своем составе имен собственных, и YARN (доля имен собственных менее 10 %).

2. Несмотря на то что YARN является проектом, основанным на методе краудсорсинга, размер его словника пока существенно меньше словника РуТез-lite. По этой причине в дальнейшем потребуются значительные временные затраты для создания полноценного ЭТ.

3. Анализ словника РуТез-lite показывает, что в нем представлено большое количество многословных выражений: больше половины от его размера. Тезаурус РуТез-lite по этому показателю существенно превосходит рассмотренные ЭТ русского языка. Эта особенность, с точки зрения авторов, обусловлена принципом его наполнения [1].

4. Число синсетов в PWN в 4—5 раз больше числа синсетов, представленных в ЭТ русского языка, однако число пар слов-синонимов в нем в 2 раза меньше, чем в РуТез-lite. Частично этот факт можно объяснить присутствием в его словнике имен собственных, которые, как правило, синонимов не имеют. Несмотря на то что словник YARN меньше словника РуТез-lite почти в 4 раза, пар синонимов в нем на данный момент всего в 2 раза меньше, чем в РуТез-lite. В то же время число синсетов в wordnet.ru почти такое же, как и в РуТез-lite, однако синонимических пар в wordnet.ru меньше на порядок (15 тыс. против 378 тыс.). Отмеченные факторы свидетельствуют о хорошей проработанности РуТез-lite, благодаря чему он в среднем содержит гораздо большее число слов в синсетах (см. рисунок).



Распределение синсетов электронных тезаурусов русского языка по числу слов

На рисунке видно, что wordnet.ru наполовину состоит из однословных синсетов, причем синсеты, состоящие из четырех слов и более, в его составе практически не представлены, в то время как в YARN и PyTez-lite таких синсетов достаточно много. Следовательно, метод перевода, которым создавался wordnet.ru, не позволяет в достаточной степени отражать отношения между словами и концепциями в языке перевода.

Анализ качества открытых ресурсов позволяет сделать вывод о том, что, несмотря на наличие большого числа разработок ЭТ русского языка, достаточно полных ресурсов, которые находятся в свободном доступе и доступны для коммерческого использования, на настоящее время нет. Таким образом, задача создания нового ЭТ русского языка, открытого и обладающего хорошей полнотой и высоким качеством данных, по-прежнему остается актуальной.

Заключение

Результаты анализа современного состояния ЭТ русского языка, представленные на основе сравнения качества, полноты и доступности данных ресурсов, позволяют сделать следующие выводы.

Существуют сложности с использованием большинства известных ЭТ русского языка, обусловленные их закрытостью, недостаточной полнотой, невозможностью использования в коммерческих целях.

Изученные ЭТ русского языка существенно уступают по своему объему английскому аналогу — ЭТ PWN.

Открытый тезаурус PyTez-lite, обладающий достаточным объемом понятий, соответствующих им слов и словосочетаний и отношений между понятиями, можно использовать в академических целях. Однако данный ЭТ обладает словником с очень большой спецификой, которая заключается в большом числе словосочетаний.

Разработка ЭТ, покрывающего достаточно полно всю общеупотребительную лексику, имеющего большое количество отношений между его понятиями и обладающего хорошим качеством данных, в настоящее время возможна только с использованием ручного труда соответствующих специалистов. При этом ручная работа делает процесс наполнения ресурса длительным, а вовлечение в него сторонних специалистов — дорогостоящим. Как следствие, авторы ЭТ не готовы выкладывать результаты своего многолетнего труда в открытый доступ, предпочитая иметь подобные ценные ресурсы в единоличном использовании.

Метод краудсорсинга для создания ЭТ требует существенных временных затрат для наполнения, что демонстрируется анализом данных проекта YARN.

Целесообразна разработка подходов, позволяющих хотя бы частично автоматизировать создание ЭТ русского языка.

Исследование выполняется при финансовой поддержке РГНФ (проект № 13-04-12020 "Новый открытый электронный тезаурус русского языка") и научной группы "Разработка методов анализа, обработки, визуализации и прогнозирования многомерных данных для современных информационных систем" Уральского федерального университета им. первого Президента России Б.Н. Ельцина.

Список литературы

1. **Лукашевич Н. В.** Тезаурусы в задачах информационного поиска. М.: Изд-во МГУ, 2011. 512 с.
2. **WordNet.** URL: http://wordnet.princeton.edu (дата обращения 02.03.2015).
3. **Орлов А. И.** Теория принятия решений. М.: Март, 2004. 656 с.
4. **Саати Т.** Принятие решений. Метод анализа иерархий. М.: Радио и связь, 1993. 278 с.
5. **Караулов Ю. Н.** Лингвистическое конструирование и тезаурус литературного языка. М.: Наука, 1981. 367 с.
6. **Морковкин В. В.** Идеографические словари. М.: Изд-во МГУ, 1970. 71 с.
7. **Баранов О. С.** Идеографический словарь русского языка. М.: ЭТС, 1995. 820 с.
8. **Русский** семантический словарь / под общей ред. Н. Ю. Шведовой. URL: http://www.slovari.ru/default.aspx?s=0&p=235
9. **Браславский П. И., Мухин М. Ю., Ляшевская О. Н.** и др. YARN: начало // Компьютерная лингвистика и интеллектуальные технологии: Тр. Междунар. конф. "Диалог-2013". М.: Изд-во РГГУ, 2013. URL: http://www.dialog-21.ru/digests/dialog2013/materials/pdf/BraslavskiyP_YARN.pdf (дата обращения: 02.03.2015).
10. **Fellbaum C.** WordNet: An Electronic Lexical Database. Cambridge, 1998. 447 p.
11. **Проект** RussNet. URL: http://project.phil.spbu.ru/RussNet/index_ru.shtml (дата обращения 02.03.2015).
12. **Азарова И. В., Митрофанова О. А., Синопальникова А. А.** Компьютерный тезаурус русского языка типа WordNet // Компьютерная лингвистика и интеллектуальные технологии: Тр. Междунар. конф. "Диалог—2003". М.: Наука, 2003. С. 43—50.
13. **Национальный** корпус русского языка. URL: http://www.ruscorpora.ru (дата обращения 02.03.2015).
14. **Азарова И. В., Митрофанова О. А., Синопальникова А. А.** и др. Разработка компьютерного тезауруса русского языка типа WordNet // Материалы конф. "Корпусная лингвистика и лингвистические базы данных"/ Под ред. А. С. Герда. СПб.: Изд-во С.-Петерб. ун-та, 2002. С. 6—19.
15. **Лингвистическая** онтология Тезаурус РуТез. URL: http://labinform.ru/pub/ruthes/ (дата обращения 02.03.2015).

16. **Добров Б. В., Лукашевич Н. В.** Тезаурус РуТез как ресурс для решения задач информационного поиска // Тр. Всеросс. конф. "Знания-Онтологии-Теории" (ЗОНТ-09). Новосибирск, 2009. С. 250—259.

17. **Сухоногов А. М., Яблонский С. А.** Разработка русского WordNet // Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Тр. 6-й Всеросс. научной конф. (RCDL'2004). Пущино, 2004. С. 113—117.

18. **Сухоногов А. М., Яблонский С. А.** Автоматизация построения англо-русского wordnet // Компьютерная лингвистика и интеллектуальные технологии: Тр. междунар. конф. "Диалог-2005" / Под ред. И. М. Кобозевой, А. С. Нариньяни, В. П. Селегея. М.: Наука, 2005, С. 25—31.

19. **Berners-Lee T., Hendler J., Lassila O.** The Semantic Web // Scientific American. 2001. Vol. 284, N. 5. P. 34—43.

20. **Гельфейнбейн И. Г., Гончарук А. В., Лехельт В. П., Липатов А. А., Шило В. В.** Автоматический перевод семантической сети wordnet на русский язык // Компьютерная лингвистика и интеллектуальные технологии: Тр. Междунар. конф. "Диалог-2003". М.: Наука, 2003. С. 193—198.

21. **Русский** WordNet. URL: http://wordnet.ru (дата обращения 02.03.2015).

22. **Yet Another** RussNet. URL: http://russianword.net (дата обращения 02.03.2015).

23. **Русский** Викисловарь. URL: http://ru.wiktionary.org (дата обращения 02.03.2015).

24. **Braslavski P., Ustalov D., Mukhin M.** A Spinning Wheel for YARN: User Interface for a Crowdsourced Thesaurus // Proc. of the Demonstrations at the 14th Conf. of the European Chapter of the Association for Computational Linguistics. Gothenburg, Sweden: Association for Computational Linguistics, 2014, pp. 101—104.

Yu. A. Kiselev, Junior Research Fellow, e-mail: yuri.kiselev@urfu.ru, **S. V. Porshnev**, Head of Department, e-mail: sergey_porshnev@mail.ru, **M. Yu. Muhkin**, Professor, e-mail: mfly@sky.ru, Ural Federal University named after the first President of Russia B. N. Yeltsin, Ekaterinburg

Current Status of Russian Electronic Thesauri: Quality, Completeness and Availability

The article discusses the problems existing in Russian electronic thesauri. The characteristic features of the most famous of them are studied. We analysed different aspects of electronic thesauri such as the quality, completeness, possibility of free use. Conclusions about the quality of thesauri are made based on the analysis of the size and composition of word lists and recall of synonymy relationships. Analysis of selected lexicographical resources and their comparison with the most famous analogue for English language discovered the need for designing methods that can allow at least partially automate the creation of Russian electronic thesaurus.

Keywords: *thesaurus, wordnet, lexical resource, Russian language*

References

1. **Lukashevich N. V.** *Tezaurusy v zadachah informacionnogo poiska* (Thesauri in application to problems of information retrieval), Moscow, Nauka MSU, 2011, 512 p. (in Russian).

2. **WordNet,** available at: http://wordnet.princeton.edu

3. **Orlov A. I.** *Teoriya prinyatiya resheniy* (Decision theory), Moscow: Izd-vo Mart, 2004, 656 p. (in Russian).

4. **Saati T.** *Prinyatie resheniy. Metod analiza ierarhiy* (Decision making. Method of hierarchy analysis), Moscow: Radio i svyaz, 1993, 278 p. (in Russian).

5. **Karaulov Ju. N.** *Lingvisticheskoe konstruirovanie i tezaurus literaturnogo jazyka* (Linguistic engineering and thesaurus of literary language), Moscow, Nauka, 1981, 367 p. (in Russian).

6. **Morkovkin V. V.** *Ideograficheskie slovari* (Ideographic dictionaries), Moscow: Izd-vo MSU, 1970, 71 p. (in Russian).

7. **Baranov O. S.** *Ideograficheskij slovar' russkogo jazyka* (Russian ideographic dictionary), Moscow: ETS, 1995, 820 p. (in Russian).

8. **Russkij** semanticheskij slovar' / Eds. N. Ju. Shvedova, available at: http://www.slovari.ru/default.aspx?s=0&p=235 (accessed 02.03.2015).

9. **Braslavskij P. I., Muhin M. Ju., Ljashevskaja O. N., Bonch-Osmolovskaja A. A., Krizhanovskij A. A., Egorov P. E.** YARN: nachalo. *Trudy Mezhdunarodnoj konferencii "Dialog-2013", 2013*, available at: http://www.dialog-21.ru/digests/dialog2013/materials/pdf/BraslavskiyP_YARN.pdf (accessed 02.03.2015).

10. **Fellbaum C.** *WordNet: An Electronic Lexical Database*. Cambridge, 1998, 447 p.

11. **Proekt** *RussNet*, available at: http://project.phil.spbu.ru/RussNet/index_ru.shtml

12. **Azarova I. V., Mitrofanova O. A., Sinopal'nikova A. A.** Komp'juternyj tezaurus russkogo jazyka tipa WordNet. *Trudy Mezhdunarodnoj konferencii "Dialog-2003",* Moscow: Nauka, 2003, pp. 43—50 (in Russian).

13. **Nacional'nyj** korpus russkogo jazyka, available at: http://www.ruscorpora.ru (in Russian).

14. **Azarova I. V., Mitrofanova O. A., Sinopal'nikova A. A., Ushakova A. A., Javorskaja M. V.** Razrabotka komp'juternogo tezaurusa russkogo jazyka tipa WordNet. *Materialy konferencii "Korpusnaja lingvistika i lingvisticheskie bazy dannyh",* St.-Petersburg: Izd-vo SpBSU, 2002, pp. 6—19 (in Russian).

15. **Lingvisticheskaja** ontologija Tezaurus RuTez, available at: http://labinform.ru/pub/ruthes/ (in Russian).

16. **Dobrov B. V., Lukashevich N. V.** Tezaurus RuTez kak resurs dlja reshenija zadach informacionnogo poiska. *Trudy Vserossijskoj konferencii "Znanija-Ontologii-Teorii" (ZONT-09),* Novosibirsk, 2009. pp. 250—259 (in Russian).

17. **Suhonogov A. M., Jablonskij S. A.** *Razrabotka russkogo WordNet*. Trudy shestoj Vserossijskoj nauchnoj konferencii RCDL'2004, Pushhino, 2004, pp. 113—117 (in Russian).

18. **Suhonogov A. M., Jablonskij S. A.** *A*vtomatizacija postroenija anglo-russkogo wordnet. *Trudy Mezhdunarodnoj konferencii "Dialog-2005"*, Moscow: Nauka, 2005, pp. 25—31 (in Russian).

19. **Berners-Lee T., Hendler J., Lassila O.** The Semantic Web. *Scientific American*, May 2001, vol. 284, no. 5, pp. 34—43.

20. **Gel'fejnbejn I. G., Goncharuk A. V., Lehel't V. P., Lipatov A. A., Shilo V. V.** Avtomaticheskij perevod semanticheskoj seti wordnet na russkij jazyk. *Trudy mezhdunarodnoj konferencii "Dialog-2003"*, Moscow, Science, 2003. (in Russian), pp. 193—198.

21. **Russkij** WordNet, available at: http://wordnet.ru

22. **Yet Another** RussNet, available at: http://russianword.net

23. **Russkij** Vikislovar', available at: http://ru.wiktionary.org

24. **Braslavski P., Ustalov D., Mukhin M.** A Spinning Wheel for YARN: User Interface for a Crowdsourced Thesaurus. *Proc. of the Demonstrations at the 14th Conf. of the European Chapter of the Association for Computational Linguistics*. Gothenburg, Sweden: Association for Computational Linguistics, 2014, pp. 101—104.

УДК 004.912:519.767.6

И. В. Трофимов, ст. науч. сотр., e-mail: itrofimov@gmail.com,
Институт программных систем им. А. К. Айламазяна РАН, г. Переславль-Залесский

Выявление упоминаний лиц в новостных текстах

Методы извлечения информации из текстов позволяют автоматически структурировать содержащуюся в документах информацию. Они находят применение в программных системах, осуществляющих обработку больших документальных массивов. В работе рассмотрена задача выявления упоминаний лиц в текстах. Исследованы возможности простых словарно-эвристических алгоритмов. Эффективность алгоритмов оценена на материале двух размеченных русскоязычных новостных коллекций.

Ключевые слова: *автоматический анализ текста, извлечение информации, распознавание именованных сущностей, выявление упоминаний лиц, словарь имен, словарь фамилий, правила извлечения информации, размеченный корпус, F-мера*

Введение

Задача извлечения информации из текстов состоит в автоматическом поиске целевой информации в электронных текстовых документах и представлении найденной информации в удобной для дальнейшего анализа структурированной форме. Ключевые отличия извлечения информации от обычного информационного поиска (*information retrieval*) состоят в следующем:

— результатом извлечения информации являются данные, а не документы;

— объектом поиска выступают целевые события, факты, объекты, а не слова.

Технически процесс извлечения информации, как правило, представляет собой автоматическое наполнение базы данных, структура которой определена заранее и описывает целевые объекты в виде, интересном с точки зрения решаемой прикладной задачи.

Извлечение информации обычно используется для обработки больших документальных массивов (обработка архивов) или систематического структурирования информации заданного типа в течение продолжительного интервала времени (веб-мониторинг, документооборот). Круг решаемых практических задач довольно разнообразен: мониторинг событий и объектов в СМИ; автоматизация обработки отзывов и жалоб в CRM-системах; создание и актуализация тематических баз данных; деперсонификация документов; связывание текстового контента и привязка контекстной рекламы; разбор библиографических записей.

Массовому практическому применению современных методов извлечения информации препятствует необходимость их тщательной настройки на конкретную прикладную задачу (высокая трудоемкость настройки). В связи с этим обстоятельством актуальным является создание инструментальных средств, упрощающих настройку. В их числе — создание “базовых” программных модулей извлечения, востребованных во многих задачах, одним из которых является модуль извлечения упоминаний лиц в тексте.

Задача извлечения предполагает нормализацию целевой информации в расчете на помещение ее во внешние по отношению к тексту структуры. Для упоминаний лиц в русском языке нормализация заключается в приведении имени к форме именительного падежа. Это довольно сложная задача, решение которой предполагает автоматическое определение словоизменительной парадигмы фамилий [1] и других компонентов имени. В то же время для ряда прикладных задач достаточно выявления (обнаружения) информации, т. е. определения мест в тексте, где упоминается целевая информация. В настоящей работе ограничимся рассмотрением только вопросов выявления упоминаний лиц в форме имени собственного.

Цель исследования — определить возможности простых словарно-эвристических алгоритмов выявления упоминаний лиц (в форме имени собственного) в русскоязычных новостных сообщениях. Алгоритмы анализируют лишь структуру имени и практически не оперируют контекстом. Оценка алгоритмов выполняется на достаточно представительных размеченных коллекциях Persons-1000 [2] и Persons-1111-F [3] (в совокупности более 2000 документов).

Современный технический уровень

К настоящему времени проведено немало исследований по распознаванию именованных сущностей (*Named Entity Recognition*, NER), в том числе и упоминаний лиц. В обзорной части работы рассмотрим лишь небольшое число публикаций, в которых представлены результаты исследований по выявлению лиц в тексте и приведены количественные оценки

качества выявления, полученные с использованием размеченной коллекции.

Уже в ранних работах по выявлению лиц в текстах были получены результаты со сбалансированной F-мерой[1], превышающей значение 90. На конференции MUC-72[2] программной системе, победившей в треке (соревновании) по автоматическому распознаванию именованных сущностей, удалось достичь значения F-меры 96 по классу сущностей person и значения 93,39 по всей совокупности именованных сущностей трека [4]. Это была гибридная система, в основе которой лежали контекстные правила распознавания и алгоритмы частичного сопоставления, опирающиеся на предварительно обученный классификатор на базе метода максимальной энтропии. Следует отметить, что NER-трек на конференции MUC-7 состоял всего из 100 новостных сообщений (язык английский). Каждая же из исследуемых нами коллекций (Persons-1000/1111-F) в 10 раз больше.

Позднее в рамках ориентированных на машинное обучение треков CoNLL[3] предлагалось решить задачу распознавания именованных сущностей алгоритмами, не зависящими от языка. В основе подхода, победившего на соревновании CoNLL-2003, лежало сопоставление результатов четырех классификаторов [5]; исследовались несколько способов их комбинирования. При выявлении лиц на тестовых множествах системе удалось получить следующие результаты: 93,85 (значение F-меры) для английского языка и 82,8 для немецкого языка. Снова отметим, что тестовые коллекции CoNLL значительно меньше [6] коллекций Persons-1000/1111-F.

Высокие результаты получены также для арабского языка. На корпусе ANERCorp (150 000 слов) было получено значение F-меры, равное 94,5, на задаче выявления лиц при помощи гибридного подхода, использующего правила и машинное обучение [7].

Для русского языка не так много работ содержат количественные оценки, полученные на каких-либо размеченных корпусах. В работах Подобряева [8, 9] описан подход к выявлению лиц в текстах на базе CRF-модели, использующей преимущественно графематические признаки, группы лексических признаков, семантические признаки[4], а также предварительную классификацию текстов для определения региона описываемых событий. На коллекциях Persons-1000/1111-F данный подход позволил получить значения F-меры, равные 84,16 и 80,91 соответственно. Близкая задача выявления именных групп, содержащих собственные имена, решалась Крейдлиным [10]. Для групп, содержащих имена лиц, на коллекции, включающей 155 таких групп, было получено значение F-меры, равное 88,8.

Возможности словаря имен

В отсутствие статистических данных было сделано предположение, что в текстах новостного жанра на русском языке для упоминания людей по имени чаще используется шаблон, который условно можно назвать Имя + Фамилия. Если это так, то при помощи словаря личных имен, учитывающего словоизменение, можно обнаружить значительную долю таких упоминаний в тексте.

Для проверки этой гипотезы использовался морфологический словарь личных имен (12 529 словарных входов) [11]. Словарь состоит преимущественно из традиционных славянских, романских, германских и других европейских имен, а также содержит небольшие подборки имен, распространенных на Кавказе, в мусульманских странах, в Средней Азии и Японии. Кроме того, в словарь были внесены имена некоторых известных современных общественно-политических деятелей из различных регионов мира.

Затем было составлено регулярное выражение (в терминах языка PSL [12, 13]), которое содержательно можно выразить следующим образом. Последовательность будем считать упоминанием лица по имени, если она состоит из следующих компонентов:

1) слово с заглавной буквы, являющееся словарным личным именем, в единственном числе;

2) одно или более кириллических слов с заглавной буквы;

3) опционально, последовательность слов в круглых скобках, в составе которой допускаются только некириллические слова, а также некоторые знаки препинания (точка, дефис, апостроф).

Первые два компонента позволяют обнаруживать как типичный новостной шаблон (например, *Иван*

---

[1] Сбалансированная F-мера определяется как среднее гармоническое между точностью и полнотой решения поисковой задачи. Точность в задаче выявления информации в тексте определяется как доля корректно выявленных целевых объектов (в нашем случае — лиц) среди всех результатов, полученных алгоритмом. Полнота — это доля выявленных алгоритмом объектов среди всего множества целевых объектов, содержащихся в обработанном текстовом массиве. Измерение значений этих величин осуществляется на эталонных текстовых коллекциях, для которых экспертами выполнена разметка всех целевых объектов. Точность и полноту часто указывают в процентах. В результате вычисленная F-мера принимает значения в диапазоне [0; 100].

[2] MUC (Message Understanding Conference) — серия конференций по автоматическому извлечению информации из текстов, проходившая в США в 1987—1998 гг. Совместно с конференциями MUC проводились мероприятия (треки) по тестированию и оценке программных систем извлечения информации о различных целевых событиях. MUC-7 — последняя конференция в серии.

[3] CoNLL (Conference on Natural Language Learning) — серия конференций по автоматической обработке текстов методами, основанными на машинном обучении. Совместно с конференцией проходит соревнование обучающихся алгоритмов. Для каждой конференции определяется так называемая общая задача и выполняется эмпирическая оценка методов решения этой задачи на размеченном корпусе текстов. В 2002—2003 гг. общей задачей было распознавание именованных сущностей.

[4] Система могла определять, является ли данное слово/именная группа упоминанием геополитической единицы или аспекта лица (должности, звания, имени, отношения родства и др.).

Таблица 1

Оценка качества решения задачи выявления лиц посредством применения регулярного выражения, основанного на словаре имен

Коллекция	F-мера	Точность	Полнота
Persons-1000	67,78	93,23	53,25
Persons-1111-F	49,72	85,97	34,97

Таблица 2

Оценка качества решения задачи выявления лиц посредством применения регулярного выражения, учитывающего элементы арабских имен

Коллекция	F-мера	Точность	Полнота
Persons-1000	68,43	93,85	53,85
Persons-1111-F	55,05	89,42	39,77

Иванов), так и ряд более редких (таких как *Иван Иванович Иванов*, *Сергей Витальевич*, *Хосе Антонио Рейес Кальдерон* и т. п.). Третий компонент предназначен для работы с упоминаниями вида *Джеймс Ставридис (James G. Stavridis*), которые согласно принятой в коллекциях разметке должны извлекаться полностью.

Применение указанного регулярного выражения позволило получить результаты, приведенные в табл. 1.

Как видно из результатов, относящихся к коллекции Persons-1000, шаблон Имя + Фамилия действительно имеет большое распространение — более 53 % упоминаний лиц по имени записаны в такой форме. Отмечаем "более", имея в виду неполноту словаря имен и недостатки использованного регулярного выражения. Например, во фразе с *сыном генпрокурора РФ Юрия Чайки Артемом Чайкой* оба упоминания не будут обнаружены. Вместо двух имен будет выявлена ошибочная последовательность из четырех слов начиная с *Юрия*.

Наиболее распространенными видами ошибок, допущенных в коллекции Persons-1000, стали следующие: соединение двух подряд идущих имен; омонимия словарных имен с другими собственными именами (например, во фразе *премьер-министр Израиля Биньямин Нетаньяху* наличие словарного имени *Израиль* приводит к выявлению более длинной цепочки слов, чем необходимо); наличие в коллекции шаблонов Фамилия + Имя + Отчество (регулярным выражением ошибочно выделяется лишь Имя + Отчество).

Результаты по коллекции Persons-1111-F существенно хуже, главным образом за счет полноты. Отмеченному факту есть несколько причин. Во-первых, недостаточная полнота словаря имен по регионам Юго-Восточной и Средней Азии. Например, индийские (хинди) индивидуальные имена могут быть составными [14], поэтому их исчерпывающее представление в словаре, видимо, невозможно. Для японских имен в СМИ наряду с транскрипционной системой Поливанова используются и другие системы[5], что также затрудняет перечисление всех возможных форм имени (ё — йо, си — ши, дзи — джи, ти — чи и т. п.). Проблема транскрипции в какой-то мере существует и для индийских имен (ья — иа, ей — ай). Во-вторых, существенна доля упоминаний, не соответствующих шаблону Имя + Фамилия. Так, например, китайские и корейские имена записывают в порядке Фамилия + Имя. При этом множество фамилий невелико, в то время как множество имен практически не ограничено. В-третьих, значительная доля текстов освещает события в странах мусульманского мира, где имена часто включают в себя артикли и другие элементы (*аль-*, *эль-*, *бин-*, *абд-* и др.), которые принято записывать со строчной буквы (иногда отдельными словами). Используя словарь таких элементов (24 элемента) и модифицировав соответствующим образом регулярное выражение, можно улучшить результат, о чем свидетельствуют данные в табл. 2.

Фамилия с инициалами

Другой распространенный шаблон для упоминания лица по имени в новостных сообщениях — Инициалы + Фамилия. Чтобы оценить долю таких случаев среди всех упоминаний, предыдущее регулярное выражение было заменено на выражение, состоящее из следующих компонентов:

1) инициал (одна заглавная кириллическая буква или *Дж*, за которыми следует точка);

2) опционально, еще один инициал, причем допускается дефис между инициалами (например, *Ж.-К. Трише*);

3) одно кириллическое слово с заглавной буквы;

4) опционально, некириллическая запись в круглых скобках (аналогично предыдущему регулярному выражению).

Полученные с помощью этого регулярного выражения результаты представлены в табл. 3.

Как видно из приведенных оценок полноты в табл. 3, отечественные журналисты избегают ис-

Таблица 3

Оценка качества решения задачи выявления лиц посредством применения регулярного выражения, учитывающего инициалы

Коллекция	F-мера	Точность	Полнота
Persons-1000	27,51	98,43	15,99
Persons-1111-F	4,26	93,94	2,18

[5] Например, при переводе англоязычных источников японские имена представляют собой кириллические кальки с латинизированной записи в системе Хэпберна.

пользования инициалов при упоминании лиц азиатского региона (за исключением России) и в то же время активно пользуются таким шаблоном в новостях, затрагивающих российский и западный мир.

Отдельные имена и фамилии

Использованные регулярные выражения (Имя + Фамилия и Инициалы + Фамилия) не должны иметь существенного пересечения по покрытию текста. По этой причине эффект их совместного применения приближенно может быть рассчитан путем суммирования полноты. Тем не менее была получена эмпирическая оценка (табл. 4).

Теперь остается открытым вопрос, что представляют собой оставшиеся 30 % полноты (в коллекции Persons-1000). Довольно легко проверить, какую долю составляют упоминания лиц в форме отдельно употребленной фамилии или имени. Достаточно применить регулярное выражение, помечающее все слова с заглавной буквы[6] как отдельные упоминания лиц, и взглянуть на полноту (учтем также арабские "префиксы"). Результат представлен в табл. 5.

Теперь, если сложить полноту по коллекции Persons-1000, становится очевидным, что доля каких-либо еще шаблонов (кроме Имя + Фамилия, Инициалы + Фамилия, отдельные имена и фамилии) не превышает 3,5 %. Отмечаем "не превышает", поскольку первые два регулярных выражения не гарантируют обнаружения всех шаблонов Имя + Фамилия и Инициалы + Фамилия.

По коллекции Persons-1111-F аналогичные выводы сделать нельзя, так как слишком велика неопределенность, обусловленная неполнотой словаря имен.

В общем случае выявление отдельных имен и фамилий нельзя отнести к задачам, эффективно решаемым простыми средствами. Для их обнаружения требуются довольно сложные шаблоны, способные учитывать разнородный (порой достаточно широкий) контекст. По мнению автора, для решения этой подзадачи целесообразно использовать методы, основанные на машинном обучении.

Тем не менее в текстах новостного жанра эта задача может решаться достаточно просто. Причина в том, что журналисты почти для каждого упоминаемого в тексте лица хотя бы раз используют форму Имя + Фамилия или Инициалы + Фамилия (обычно в момент интродукции лица в тексте). Учитывая эту особенность жанра, был разработан простой программный модуль, который составлял словарь фамилий, ранее обнаруженных в тексте регулярным выражением для Имя + Фамилия[7] (фамилией считалось последнее слово, обнаруженное регулярным выражением). Для каждого анализируемого текста составлялся свой индивидуальный словарь из обнаруженных в этом тексте фамилий. Затем для выявления фамилий, упомянутых без имени или инициалов, использовалось частичное сопоставление по этому словарю. При сопоставлении проверялось каждое слово текста, начинающееся с заглавной буквы. У алгоритма частичного сопоставления были выделены два параметра:

- усечение; определяет, сколько символов с конца слова можно не рассматривать при сопоставлении;
- минимальная длина слова, при котором включается усечение; если длина проверяемого слова меньше, то сопоставление считается успешным только при условии точного совпадения текстовой и словарной формы.

Эмпирически были подобраны оптимальные значения этих параметров по коллекции Persons-1000 (табл. 6).

В табл. 6 отражена зависимость значения F-меры (при совместном использовании шаблонов Имя + Фамилия, Инициалы + Фамилия и алгоритма частичного сопоставления) от параметров алгоритма частичного сопоставления. В скобках указаны точность и полнота соответственно. Максимальные значения каждого из параметров выделены полужирным шрифтом.

Из данных табл. 6 становится понятно, что из 26 % полноты (в коллекции Person-1000), приходящихся на отдельно стоящие имена и фамилии, более 24 % составляют фамилии — максимальная достигнутая полнота (94,56) минус полнота без алгоритма частичного сопоставления (69,84).

Попытки дополнительно обнаружить отдельные фамилии словарными методами не привели к существенным изменениям результатов. В качестве словарных ресурсов использовались морфологический

Таблица 4

Эффект работы метода при совместном использовании регулярных выражений Имя + Фамилия и Инициалы + Фамилия

Коллекция	F-мера	Точность	Полнота
Persons-1000	80,45	94,86	69,84
Persons-1111-F	57,15	89,64	41,95

Таблица 5

Доля упоминаний лиц по имени в форме отдельных употреблений имени или фамилии

Коллекция	Полнота
Persons-1000	26,65
Persons-1111-F	35,78

[6] Будем полагать, что употребления фамилий вида *де Гроот*, *фон Беттихер* и т. д. довольно редки.

[7] Брать фамилии из шаблона Инициалы + Фамилия из-за неполноты словаря имен рискованно. Например, если в тексте встречаются личные имена *Наото Кан* и *Н. Кан*, а в словаре имен нет *Наото*, то после обнаружения фамилии *Кан* в конструкции с инициалами она будет выявлена и после *Наото*, что приведет к ошибке, так как имя останется невыявленным.

Таблица 6

Зависимость значения F-меры от параметров алгоритма частичного сопоставления

Минимальная длина слова для усечения (в символах)	Усечение (в символах)			
	1	2	3	4
3	94,21 (95,07/93,37)	94,61 (94,89/94,33)	—	—
4	94,21 (**95,10**/93,33)	**94,64** (94,99/94,29)	94,54 (94,53/**94,56**)	—
5	94,16 (95,09/93,24)	94,59 (94,99/94,19)	94,59 (94,72/94,45)	94,02 (93,55/94,50)
6	93,86 (95,08/92,66)	94,28 (95,00/93,57)	94,36 (94,90/93,83)	94,21 (94,55/93,87)

словарь частотных русских фамилий [15], а также аналогичный словарь, составленный для лиц, часто фигурирующих в СМИ (были использованы списки "Персон года" за последние три года, публикуемые на сайте компании "Медиалогия", — 146 различных фамилий). Увеличение полноты выявления составляло доли процента.

Кроме того, предпринималась попытка применить метод частичного сопоставления для выявления отдельных имен (словарь имен составлялся первым регулярным выражением). Такой метод также не позволил улучшить значение F-меры, а рост полноты составлял доли процента.

Результаты совместного применения вышеупомянутых регулярных выражений и алгоритма частичного сопоставления для фамилий приведены в табл. 7.

Словари китайских и корейских фамилий

Как отмечалось ранее, китайские и корейские имена принято записывать согласно шаблону Фамилия + Имя, причем перечень фамилий невелик. Чтобы выявить в тексте упоминания таких имен, были составлены словари наиболее распространенных китайских и корейских фамилий (125 и 181 словарных входов соответственно). Затем использовались следующие регулярные выражения. Для китайских имен:

1) слово с заглавной буквы, являющееся китайской фамилией;

Таблица 7

Результаты совместного применения регулярных выражений Имя + Фамилия, Инициалы + Фамилия и алгоритма частичного сопоставления на базе динамически формируемого словаря фамилий

Коллекция	F-мера	Точность	Полнота
Persons-1000	94,64	94,99	94,29
Persons-1111-F	69,69	87,67	57,83

Таблица 8

Оценка качества выявления лиц после применения регулярных выражений, работающих со словарями китайских и корейских фамилий

Коллекция	F-мера	Точность	Полнота
Persons-1000	94,82	95,04	94,60
Persons-1111-F	72,76	88,52	61,76

2) кириллическое слово с заглавной буквы, содержащее не более четырех гласных;

3) опционально, некириллическая запись в круглых скобках.

Для корейских имен:

1) слово с заглавной буквы, являющееся корейской фамилией;

2) два кириллических слова с заглавной буквы, содержащих не более двух гласных и длиной не более пяти символов;

3) опционально, некириллическая запись в круглых скобках.

Затем исключались случаи, когда фамилия длиной более одного символа полностью записана заглавными буквами, так как это приводит к смешению с аббревиатурами (короткие фамилии). Также исключались случаи, когда фамилия стоит первой в предложении и совпадает с русскоязычным предлогом, частицей, местоимением или союзом.

Работа этих правил позволила лишь немного улучшить предыдущий результат (табл. 8).

Мелкие детали

В попытках улучшить результат были написаны еще несколько регулярных выражений, нацеленных на обработку различных частных случаев. Так, например, в европейских личных именах встречаются компоненты, которые исторически указывали на местность (*ван*, *фон*, *де* и т. п.), а впоследствии трансформировались в часть фамилии. Графематически они записываются со строчной буквы. Был составлен

Таблица 9

Оценка качества выявления лиц всей совокупностью эвристик

Коллекция	F-мера	Точность	Полнота
Persons-1000	96,96	98,68	95,29
Persons-1111-F	73,71	90,86	62,01

словарь такого рода компонентов и модифицировано регулярное выражения для Имя + Фамилия, чтобы они могли быть учтены.

Следующим шагом была работа с шаблоном Фамилия + Имя + Отчество для ограниченного списка характерных окончаний фамилий и отчеств. Регулярное выражение требует наличия последовательности из следующих компонентов:

1) слово с заглавной буквы, заканчивающееся на *-ов/-ова*, *-ев/-ева*, *-ин/-ина*, *-ский/-ская*, *-цкий/цкая*, *-нко*, *-швили*, *-дзе* (с учетом словоизменения при склонении);

2) слово, являющееся словарным именем;

3) слово с заглавной буквы, заканчивающееся на *-вич/-вна* (также с учетом словоизменения при склонении).

Вероятно, списки характерных окончаний можно расширить.

Кроме этого, были предприняты шаги для повышения точности выявления за счет исключения из словаря "проблемных" имен, омонимичных другим объектам (*Израиль*, *Банка*, *Ливия*, *Рада* и т. д.), и составления регулярного выражения, запрещающего выявлять конструкции вида *имени А. С. Пушкина*, которые не должны выявляться согласно инструкции к коллекции Persons-1000/1111-F.

В результате удалось добиться представленных в табл. 9 показателей качества.

Заключение

Описанный словарно-эвристический подход к выявлению упоминаний лиц в текстах новостных сообщений позволяет добиться значения F-меры, превышающего 95 (для упоминаний с европейской структурой имени). Можно предположить, что дальнейшее наполнение словаря имен и списков исключений (для имен, омонимичных другим объектам) позволит получить сопоставимый результат для лиц из ряда других регионов мира без необходимости модификации алгоритмов. В то же время нельзя утверждать, что такой бесконтекстный подход окажется работоспособен на текстах неновостного жанра.

Подход может найти самостоятельное применение в задачах обогащения гипертекстового контента (связывание контента, семантическое обогащение, стилевое выделение упоминаний лиц), индексирования документов по упоминаемым лицам, а также в рамках комплексных систем текстовой аналитики в качестве модуля.

Работа выполнялась в рамках проекта RFMEFI60414X0138 при финансовой поддержке государства в лице Минобрнауки России.

Список литературы

1. **Сулейманова Е. А., Константинов К. А.** Об эвристическом методе разрешения неоднозначности при морфологическом анализе незнакомых фамилий // Машинное обучение и анализ данных. 2013. Т. 1, № 5. С. 519—525.

2. **Коллекция** Persons-1000 [Электронный ресурс]. URL: http://ai-center.botik.ru/Airec/index.php/ru/collections/28-persons-1000 (дата обращения 26.01.2015).

3. **Коллекция** Persons-1111-F [Электронный ресурс]. URL: http://ai-center.botik.ru/Airec/index.php/ru/collections/29-persons-1111-f (дата обращения 26.01.2015).

4. **Mikheev A., Grover C., Moens M.** Description of the LTG System Used for MUC-7 // In Proceedings of the Seventh Message Understanding Conference. Fairfax, Virginia, 29 April — 1 May, 1998. URL: http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_proceedings/ltg_muc7.pdf (дата обращения 26.01.2015).

5. **Radu F., Abe I., Hongyan J., Tong Z.** Named Entity Recognition through Classifier Combination // Proceedings of CoNLL-2003. Edmonton, Canada, 2003, pp. 168—171.

6. **Tjong Kim Sang E. F., De Meulder F.** Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition // Proceedings of CoNLL-2003. Edmonton, Canada, 2003. P. 142—147.

7. **Mai O., Khaled S.** Person name recognition using the hybrid approach // Natural Language Processing and Information Systems, volume 7934 of Lecture Notes in Computer Science. Berlin Heidelberg: Springer. 2013. P. 237—248.

8. **Подобряев А. В.** Поиск упоминаний лиц в новостных текстах с использованием CRF модели // Тр. XV Всеросс. науч. конф. RCDL'2013 "Электронные библиотеки: перспективные методы и технологии, электронные коллекции". Ярославль, 14—17 октября 2013 г. URL: http://rcdl2013.uniyar.ac.ru/doc/full_text/s10_1.pdf (дата обращения 26.01.2015).

9. **Подобряев А. В.** Региональный классификатор текстов для поиска упоминаний лиц в новостных текстах // Тр. XVI Всеросс. науч. конф. RCDL'2014 "Электронные библиотеки: перспективные методы и технологии, электронные коллекции". Дубна, 13—16 октября 2014 г. Дубна: ОИЯИ, 2014. С. 214—216.

10. **Крейдлин Л. Г.** Программа выделения русских индивидуализированных именных групп TAGLITE // Компьютерная лингвистика и интеллектуальные технологии (Диалог-2005). Тр. Междунар. конф. Звенигород, 1—6 мая 2005 г. М.: Наука, 2005. С. 292—297.

11. **Морфологический** словарь личных имен [Электронный ресурс]. URL: http://ai-center.botik.ru/Airec/index.php/ru/resources/dictionaries/35-persons-morpho-dictionary (дата обращения 12.02.2015).

12. **Кормалев Д. А., Куршев Е. П., Сулейманова Е. А., Трофимов И. В.** Технология извлечения информации из текстов, основанная на знаниях // Программные продукты и системы. 2009. Т. 86, № 2. С. 62—66.

13. **Александровский Д. А., Кормалев Д. А., Кормалева М. С.** и др. Развитие средств аналитической обработки текста в системе ИСИДА-Т // Тр. Десятой нац. конф. по искусственному интеллекту с междунар. участием КИИ-2006, Обнинск, 25—28 сентября 2006 г.: В 3 т. М.: Физматлит, 2006. С. 555—563.

14. **Системы** личных имен у народов мира. М.: Главная редакция восточной литературы издательства "Наука", 1989. 383 с.

15. **Журавлев А. Ф.** К статистике русских фамилий // Вопросы ономастики. 2005. Т. 1. № 2. С. 126—146.

I. V. Trofimov, Senior Research Assistant, e-mail: itrofimov@gmail.com, Ailamazyan Program Systems Institute of Russian Academy of Sciences, Pereslavl-Zalessky

Person Name Recognition in Newswire Text

Named entity recognition is an important part of present-day natural language processing. The paper is concerned with the problem of recognizing person mentions (in the form of proper names) in Russian-language newswire text. The goal of research is to determine what F-measure rates can be achieved for Russian with fairly simple dictionary-heuristic methods, the results serving as a starting point for further research. The method is based on the use of person first-name morphological dictionary (more than 12,000 names) and regular expressions describing name structure independently of context. The method was evaluated on a Russian-language newswire text corpus (more than 2,000 documents). The study showed that two simple regular expressions for full-name recognition, along with a procedure recognizing stand-alone surname reoccurrence, can yield an F-measure score in access of 94, for European names. As for non-European names, the F-measure value never came close to 80 (because of low recall), in spite of the fact that we used lists of common Muslim name parts, Chinese and Korean surname vocabulary, and took into account the inverted order (family name followed by the given name) typical of Chinese and Korean person names. Low recall is due to incompleteness of first-name vocabulary, inconsistent name transliteration, and some other factors. We believe that it is still possible to achieve higher F-measure scores (especially for Asian names) through mere vocabulary extension.

The proposed approach can find application in tasks like hypertext content enrichment (content linking, semantic enrichment, name highlighting), person-based document indexing, or be used as part of an integrated text analysis system.

Keywords: *natural language processing, information extraction, named entity recognition, person name recognition, vocabulary of first names, vocabulary of family names, information extraction rules, annotated corpus, F-measure.*

References

1. **Suleymanova E. A., Konstantinov K. A.** Ob evristicheskom metode razresheniya neodnoznachnosti pri morfologicheskom analize neznakomyh familij. *Mashinnoe obuchenie i analiz dannyh*, 2013, vol. 1, no. 5, pp. 519—525 (in Russian).

2. **Kollekciya** Persons-1000, available at: http://ai-center.botik.ru/Airec/index.php/ru/collections/28-persons-1000 (accessed 26.01.2015).

3. **Kollekciya** Persons-1111-F, available at: http://ai-center.botik.ru/Airec/index.php/ru/collections/29-persons-1111-f (accessed 26.01.2015).

4. **Mikheev A., Grover C., Moens M.** Description of the LTG System Used for MUC-7. *In Proceedings of the Seventh Message Understanding Conference.* Fairfax, Virginia, 29 April — 1 May, 1998, available at: http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_proceedings/ltg_muc7.pdf

5. **Radu F., Abe I., Hongyan J., Tong Z**. Named Entity Recognition through Classifier Combination. *Proceedings of CoNLL-2003.* Edmonton, Canada, 2003, pp. 168—171.

6. **Tjong Kim Sang E. F., De Meulder F.** Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *Proceedings of CoNLL-2003.* Edmonton, Canada, 2003, pp. 142—147.

7. **Mai O., Khaled S.** Person name recognition using the hybrid approach. *Natural Language Processing and Information Systems, vol. 7934 of Lecture Notes in Computer Science*. Springer, Berlin Heidelberg, 2013, pp 237—248.

8. **Podobryaev A. V.** Poisk upominanij lic v novostnyh tekstah s ispol'zovaniem CRF modeli. *Trudy XV Vserossijskoj nauchnoj konferencii RCDL'2013 "Ehlektronnye biblioteki: perspektivnye metody i tekhnologii, ehlektronnye kollekcii"*. Yaroslavl', 2013, available at: http://rcdl2013.uniyar.ac.ru/doc/full_text/s10_1.pdf (in Russian).

9. **Podobryaev A. V.** Regional'nyj klassifikator tekstov dlya poiska upominanij lic v novostnyh tekstah. *Trudy XVI Vserossijskoj nauchnoj konferencii RCDL-2014 "Ehlektronnye biblioteki: perspektivnye metody i tekhnologii, ehlektronnye kollekcii"*. Dubna, 13—16 oktyabrya 2014. Dubna: OIYAI, 2014, pp. 214—216 (in Russian).

10. **Krejdlin L. G.** Programma vydeleniya russkih individualizirovannyh imennyh grupp TAGLITE. *Komp'yuternaya lingvistika i intellektual'nye tekhnologii (Dialog'2005). Trudy mezhdunarodnoj konferencii*. Zvenigorod, 1—6 May, 2005. Moscow: Nauka, 2005, pp. 292—297 (in Russian).

11. **Morfologicheskij** slovar' lichnyh imen, available at: http://ai-center.botik.ru/Airec/index.php/ru/resources/dictionaries/35-persons-morpho-dictionary (accessed 12.02.2015).

12. **Kormalev D. A., Kurshev E. P., Sulejmanova E. A., Trofimov I. V.** Tekhnologiya izvlecheniya informacii iz tekstov, osnovannaya na znaniyah. *Programmnye produkty i sistemy,* 2009, vol. 86, no. 2, pp. 62—66 (in Russian).

13. **Aleksandrovskij D. A., Kormalev D. A., Kormaleva M. S., Kurshev E. P., Sulejmanova E. A., Trofimov I. V.** Razvitie sredstv analiticheskoj obrabotki teksta v sisteme ISIDA-T. *Trudy Desyatoj nac. konf. po iskusstvennomu intellektu s mezhdunarodnym uchastiem KII-2006*, Obninsk, 25—28 September, 2006. Moscow: Fizmatlit, 2006, pp. 555—563 (in Russian).

14. **Sistemy** *lichnyh imen u narodov mira*. Moscow: Glavnaya redakciya vostochnoj literatury izdatel'stva "Nauka", 1989 (in Russian).

15. **Zhuravlev A. F.** K statistike russkih familij. *Voprosy onomastiki,* 2005, vol. 1, no. 2, pp. 126—146 (in Russian).

Российский фонд фундаментальных исследований
Институт атомной энергетики университета МИФИ
Институт проблем информатики РАН
Московская секция ACM SIGMOD

XVII Международная конференция DAMDID/RCDL'2015

"Аналитика и управление данными в областях с интенсивным использованием данных"

13—16 октября 2015 года, г. Обнинск, Россия

http://www.damdid2015.iate.obninsk.ru

Конференция "Аналитика и управление данными в областях с интенсивным использованием данных" ("Data Analytics and Management in Data Intensive Domains" (DAMDID)) планируется как междисциплинарный форум исследователей и практиков из разнообразных областей, содействующий сотрудничеству и обмену идеями в сфере анализа и управления данными в условиях их интенсивного использования (data intensive domains (DID)).

Конференция DAMDID образуется в результате трансформации конференции RCDL (http://rcdl.ru) с сохранением преемственности по отношению к RCDL после 16 лет ее успешного функционирования.

Для участия в конференции приглашаются специалисты из таких областей, как X-информатика (где в качестве X выступают астро-, био-, гео-, нейро-, медицина, физика, химия и др.), социальные науки, экономика, финансы и др., а также специалисты из областей статистики, машинного обучения, data mining, data science, новых технологий и ИТ, бизнеса и др.

В отличие от других конференций DAMDID/RCDL акцентирует рассмотрение проблем управления данными и извлечения знаний из данных в процессе создания конкретных массивных коллекций данных и решения конкретных задач на таких коллекциях. Ожидается, что взаимное дополнение подходов в междисциплинарных DID будет способствовать развитию корпоративной культуры, обобщающей методы анализа данных и создания информационных систем, применяемых в различных DID.

В рамках конференции проводится Диссертационный семинар, ориентированный на молодых исследователей.

Труды конференции будут опубликованы в виде сборника текстов принятых полных статей, кратких статей и тезисов стендовых докладов. Тексты полных статей будут также представлены для опубликования в европейский репозиторий трудов конференций CEUR Workshop Proceedings. Лучшие статьи, представленные на конференции, будут рекомендованы к публикации в изданиях, признанных ВАК, таких как "Информатика и ее применения", "Программная инженерия", "Системы и средства информатики".

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *З. В. Наумаова*

Сдано в набор 07.03.2015 г. Подписано в печать 20.05.2015 г. Формат 60×88 1/8. Заказ PI615
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д.19, стр.1. Сайт: **www.aov.ru**