УДК 004.057.2

Д.В. Силаков, аспирант,
Институт Системного Программирования РАН,
E-mail: silakov@ispras.ru

Информационно-аналитическая система для разработки и использования Базового стандарта ОС Linux (LSB)

Предложена логическая модель системы интерфейсов приложений с ОС Linux и основанный на этой модели метод построения и развития интерфейсных стандартов ОС Linux, нацеленных на обеспечение возможности создания приложений, переносимых между различными дистрибутивами Linux.

Проблема разработки переносимых приложений для ОС Linux

В последние годы наблюдается неуклонный рост популярности операционной системы Linux, которая все больше используется не только энтузиастами, но и крупными коммерческими компаниями и государственными учреждениями. Тем не менее, доля Linux во многих сегментах рынка (в частности, на настольных компьютерах) все еще мала. Одной из основных причин, препятствующих увеличению доли Linux в таких сегментах, является отсутствие в этой операционной системе необходимых пользователям приложений. В свою очередь, одной из причин отсутствия приложений является разнообразие существующих операционных систем, основанных на ядре Linux (и называемых дистрибутивами Linux). Так, по данным [1], существует более пятисот дистрибутивов Linux, поддерживаемых в настоящее время – и это без учета узкоспециализированных систем и систем, создаваемых энтузиастами.

Помимо общего ядра, в основе большинства дистрибутивов Linux лежит схожий набор компонентов (библиотек, утилит, средств разработки, прикладных программ), создаваемых сторонними разработчиками. Многие

разработчики в мире свободного ПО придерживаются принципа “Выпускайте версии рано, выпускайте версии часто” [2] – обновления для компонентов могут выпускаться по несколько раз в месяц, что порождает большое количество различных версий одного и того же компонента. При этом по функциональности каждая новая версия может отличаться от предыдущей. Кроме того, разработчики дистрибутивов часто модифицируют существующие версии компонентов при добавлении их в свои системы – например, с целью предоставления новой функциональности, которая будет выгодно выделять их продукт среди прочих. В результате функциональность одного и того же компонента в разных дистрибутивах может сильно отличаться.

Разнообразие дистрибутивов предоставляет богатый выбор пользователям, однако осложняет создание программ, переносимых между различными системами – разработчикам приложений необходимо удостовериться, что все используемые библиотеки и их интерфейсы присутствуют во всех целевых системах и имеют там одинаковую функциональность.

Используемые разработчиками приложений методы увеличения числа дистрибутивов, в которых может корректно функционировать их продукт, зависят от способа распространения программы. Различают открытое ПО, исходный код которого доступен для просмотра и изменения третьим лицам, а также проприетарное ПО, монопольное право на модификацию которого принадлежит правообладателям.

Разработчики приложений с открытым исходным кодом обычно полагаются на разработчиков дистрибутивов, которые включают их программы в свои системы – как правило, в каждом дистрибутиве есть специалист, ответственный за корректное функционирование конкретного приложения в дистрибутиве, который может адаптировать код приложения в случае необходимости. Однако отметим, что чем больше изменений

приходиться вносить, тем больше вероятность, что полученная в результате модификаций программа будет сильно отличаться от оригинала и не отражать задумок авторов исходного продукта.

Разработчики проприетарных продуктов, не предоставляющие исходный код, не могут полагаться на производителей ОС и должны обеспечивать совместимость своих продуктов с каждой целевой системой на бинарном уровне – пользователям предоставляются только двоичные файлы программы, и задачей разработчиков приложения является обеспечение корректного функционирования этих файлов в каждом дистрибутиве. Однако сборка и тестирование приложения на каждом существующем дистрибутиве не являются экономически целесообразными для разработчиков ПО. Поэтому многие производители проприетарного ПО поддерживают лишь ограниченное число дистрибутивов Linux – как правило, в качестве целевых рассматриваются системы, которым принадлежит существенная часть рынка – в частности, SUSE Enterprise Linux и Red Hat Enterprise Linux. Однако конечные пользователи ожидают увидеть продукт “для Linux”, а не “для SUSE” или “для Red Hat”.

Одним из подходов к облегчению разработки приложений, совместимых с различными дистрибутивами, является стандартизация – выделение требований, которым должны удовлетворять все реализации ОС, соответствующие стандарту. Для разработчиков приложений важна гарантия наличия в системе определенных библиотек и функций.

Количество функций-кандидатов на включение в стандарт может быть велико – современный дистрибутив Linux, поставляющийся на одном DVD-диске, содержит несколько тысяч библиотек, экспортирующих сотни тысяч функций. Стандартизация – достаточно дорогой и трудоемкий процесс, включающий в себя, как правило, анализ существующих реализаций, разработку документации и тестов. Поэтому важно уметь оценивать реальные потребности приложений и возможности

существующих систем, чтобы в первую очередь включать в стандарт наиболее востребованные элементы.

Ввиду большого количества потенциальных объектов стандартизации, размеры стандартов могут оказаться достаточно большими, что может затруднить их использование разработчиками. Облегчить создание ПО, отвечающего требованиям того или иного стандарта, может наличие различного рода инструментов, использование которых в процессе разработки гарантирует соблюдение таких требований. К таким средствам можно отнести тестовый набор, проверяющий приложение на соответствие стандарту. Для поддержки создания приложений, отвечающих требованиям стандарта, может предоставляться специальная среда сборки. Все такие вспомогательные компоненты, образующие окружение стандарта, должны поддерживаться в согласованном состоянии с текстом стандарта.

Важным аспектом стандартизации является разработка профилей – объединение существующих стандартов либо их подмножеств с целью создания спецификации, охватывающей определенный класс целевых систем, каждая отдельная составляющая которых уже рассмотрена в одном из имеющихся стандартов. В области разработки ПО подобные спецификации востребованы, в частности, при создании узкоспециализированных продуктов – например, предназначенных только для работы на сервере либо для работы в мобильных устройствах. Производителям таких приложений, как правило, интересны только операционные системы, работающие на целевых платформах, и для них полезным является наличие стандарта, описывающего только такой класс систем. При наличии одного или нескольких стандартов, уже охватывающих необходимую область, создание требуемой спецификации может быть существенно упрощено и ускорено за счет использования готовых решений. Однако при выделении подмножеств стандартов, равно

как и при их объединении, встает задача обеспечения непротиворечивости и согласованности получаемого профиля.

Одним из наиболее известных интерфейсных стандартов для операционных систем является Единая спецификация UNIX (Single Unix Specification, SUS), в основе которой лежит стандарт POSIX, созданный для обеспечения переносимости прикладных программ между различными UNIX-подобными системами на уровне исходного кода. При таком подходе стандартизации подлежит Интерфейс программирования приложений (API, Application Programming Interface), основную часть которого составляют функции, предоставляемые заголовочными файлами системы. Гарантируется возможность перекомпиляции приложения из исходного кода в каждой совместимой со стандартом ОС. Альтернативой является стандартизация бинарного интерфейса приложений (Application Binary Interface, ABI) обеспечивающая возможность использования одних и тех же файлов приложения во всех совместимых ОС, без необходимости перекомпиляции. Основным объектом стандартизации при этом являются бинарные файлы библиотек, предоставляемые ОС, и экспортируемые ими сущности, называемые бинарными символами. Такой подход в настоящее время используется Базовым стандартом Linux (Linux Standard Base, LSB).

Истоки SUS восходят к спецификации Common API Specification, разработанной в начале 1990х годов альянсом COSE, в состав которого входили все ведущие производители UNIX-систем того времени. Основной целью альянса было выявление интерфейсов, общих для существовавших на тот момент реализаций UNIX. Результирующий список содержал 1170 интерфейсов (по этой причине документ также известен как Spec 1170). В 1992-1993 годах, уже в рамках разработки SUS, было дополнительно исследовано пятьдесят ведущих на тот момент приложений для UNIX и выявлено 130 функций-кандидатов на включение в стандарт [3].

Процесс анализа приложений в ходе разработки SUS и Spec 1170

производился практически без автоматизации и сводился к изучению исходного кода аналитиками. В начале 1990х годов такой подход являлся удовлетворительным и позволил добиться качественного и достаточно полного анализа. Однако к настоящему времени количество объектов для анализа, а также их размер существенно увеличились – так, при разработке Базового стандарта Linux производится анализ десятков дистрибутивов и тысяч приложений Linux, рассматриваются сотни библиотек и сотни тысяч интерфейсов. Объем стандарта LSB также существенно превосходит POSIX – последняя версия, LSB 4.0, специфицирует около 38.000 интерфейсов. Такие объемы данных приводят к необходимости автоматизации многих процессов, связанных с разработкой и развитием стандарта.

Итак, можно сформулировать следующие задачи, стоящие перед информационно-аналитической системой для обеспечения возможности создания приложений, переносимых между различными дистрибутивами Linux:

- обеспечение проверки приложений и дистрибутивов на совместимость требованиям стандарта;
- планирование развития стандарта;
- создание новых версий стандарта и его профилей.

Метод развития интерфейсных стандартов ОС Linux

Метод развития интерфейсных стандартов ОС Linux, предлагаемый в данной работе, включает следующие этапы:

- Проведение анализа экосистемы Linux
  - выделение популярных приложений, анализ их требований к библиотекам;
  - сбор информации об основных дистрибутивах

Состав и свойства приложений и дистрибутивов постоянно меняются

за счет выхода новых версий, поэтому необходимо иметь информацию не об одномоментном состоянии экосистемы, а процессе ее эволюции в течение нескольких последних лет. Должен осуществляться постоянный мониторинг экосистемы Linux, а его результаты, собранные к определенным моментам времени, могут быть использованы для создания очередной версии стандарта, как показано на Рисунке 1.

- Подготовка новой версии стандарта. Данный этап включает выбор интерфейсов, наиболее востребованных приложениями, и предоставляемых дистрибутивами, и построение на основе выделенного списка согласованного замыкания интерфейсов, которые будут включены в стандарт.
- Добавление семантической информации об интерфейсах (в частности, описание предоставляемой функциональности), разработка тестов и внесение необходимых модификаций в систему сертификации на соответствие стандарту.

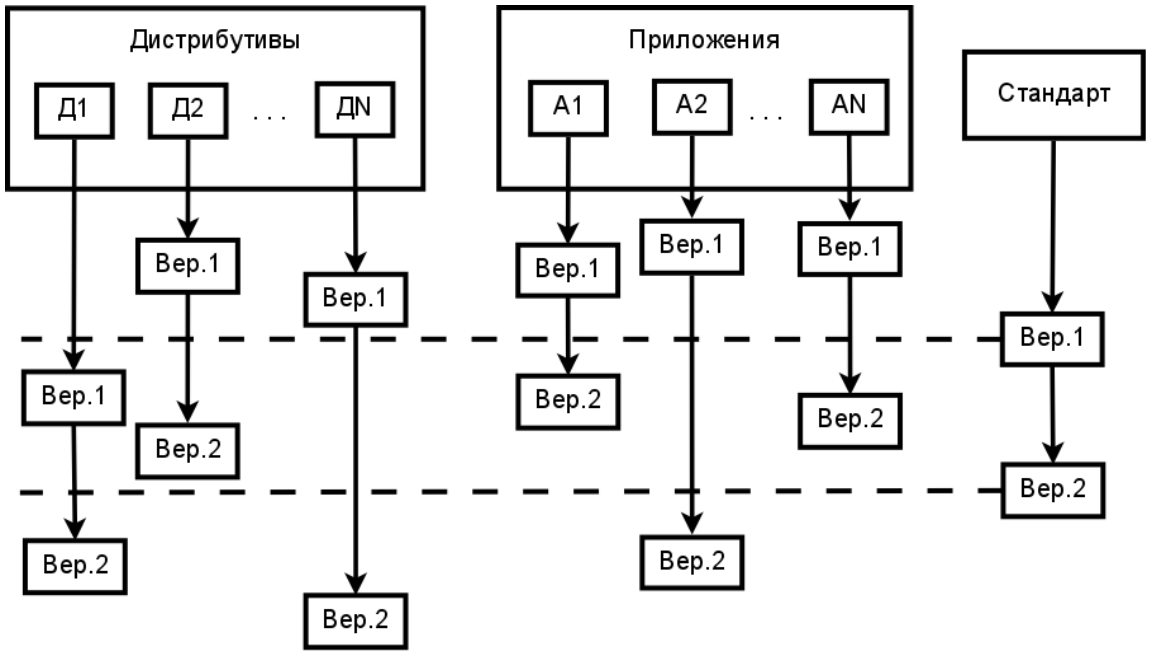

Рисунок 1. Анализ экосистемы Linux при разработке стандарта.

Предлагаемый процесс опирается на информационную систему,

которую можно рассматривать как логическую модель интерфейсов экосистемы Linux. Технологии построения этой модели посвящена основная часть этой статьи.

Логическая модель системы интерфейсов приложений с ОС Linux

В Базовом стандарте Linux основным объектом стандартизации является бинарный интерфейс приложений. Поэтому в работе рассматриваются приложения, поставляемые в виде бинарных файлов – исполняемых файлов и разделяемых библиотек. Основным форматом для таких файлов библиотек в ОС Linux является ELF (“Executable and Linking Format”). Общее описание формата приведено в спецификации System V ABI [4]; дополнения, специфичные для Linux-систем, описаны в соответствующем разделе стандарта LSB [5].

Описываемые стандартом характеристики интерфейсов можно разделить на две группы:

- структурные характеристики, которые могут быть проверены статически – например, состав библиотек или сигнатуры функций;
- семантические характеристики, для проверки которых в общем случае необходимо проведение тестирования – например, поведение функций.

Модель, предлагаемая в данной работе, охватывает структурное описание интерфейсов, абстрагируясь от семантических аспектов. В качестве объектов хранения в модели представлены интерфейсы, задействованные в процессе работы динамического загрузчика системы [6]. Совместимость приложения и дистрибутива по таким интерфейсам позволяет гарантировать успешность запуска приложения в дистрибутиве.

Выделены следующие виды интерфейсов:

- библиотеки – специальный вид файлов формата ELF, которые могут экспортировать интерфейсы;

- бинарные символы, экспортируемые библиотеками – сущности бинарного уровня, соответствующие функциям и глобальным переменным;
- структура и размер типов, используемых в качестве параметров экспортируемых функций и их возвращаемых значений;
- атрибуты ELF файлов, участвующих в процессе связывания: разрядность, целевая аппаратная архитектура и типы секций, присутствующих в файле.

Предложенная модель не учитывает следующие способы взаимодействия приложений с ОС Linux:

- динамическая загрузка библиотек и обращение к экспортируемым ими символам в процессе работы программы (например, с использованием функций библиотеки “dlopen”);
- вызов внешних команд и утилит в процессе работы программы (например, посредством функций “system” или “exec”).

Однако в современных рекомендациях по разработке переносимых программ использование подобных возможностей считается приемлемым только по отношению файлов, являющихся частью приложения. Вовлекать же в процесс взаимодействия с ОС любой файл, в наличии которого разработчики не могут быть уверены, считается плохой практикой [7], поскольку может привести к аварийному завершению работы программы.

Информационная система для поддержки развития и использования интерфейсных стандартов ОС Linux

При реализации рассматриваемого метода развития интерфейсных стандартов ОС Linux в данной работе предлагается использовать информационно-аналитическую систему, состоящую из следующих компонентов:

- база данных, содержащая сведения как о стандартизированных интерфейсах, так и об интерфейсах, используемых существующими

приложениями и предоставляемых дистрибутивами. Схема базы основывается на модели системы интерфейсов, рассмотренной ранее.

- инструменты по сбору информации для заполнения базы. Ввиду большого количества существующих продуктов и потенциальных кандидатов на стандартизацию, ручной сбор необходимых данных может оказаться малоэффективным, поэтому необходима автоматизация этого процесса;
- инструменты, использующие базу данных для создания компонентов окружения стандарта.

В базе данных должна храниться информация обо всех интерфейсах и их параметрах, входящих в стандарт и используемых хотя бы одним из компонентов окружения. Если при создании некоторого компонента возникает необходимость узнать список объектов определенного вида, которые включены в стандарт, а также их свойства, установленные стандартом, то эта информация должна извлекаться из базы данных. Такой подход обеспечивает согласованность компонентов относительно стандартизированных объектов. При этом необходимо, чтобы информация в базе была синхронизирована с текстом стандарта. Одним из методов обеспечения такой синхронизации является генерация частей текста с использованием базы данных – в таком случае база становится единственным первичным источником информации о стандартизованных объектах.

Помимо сведений о стандартизованных сущностях, для обеспечения согласованности компонентов друг с другом полезно хранить в базе данные, используемые более чем одним компонентом, даже если эти данные не имеют прямого отношения к самому стандарту.

Для хранения информации об элементах существующих дистрибутивов и приложений в данной работе предлагается каждый вид

интерфейсов представлять в схеме базы данных двумя сущностями – одна соответствует интерфейсам данного вида, входящим в стандарт, другая – интерфейсам, присутствующим и используемым в реальных системах. Такой подход представляется целесообразным, поскольку данные о стандартизованных объектах и о существующих системах используются с разными целями – для создания компонентов окружения стандарта и для принятия решений – и в общем случае указанные два типа сущностей должны обладать разным набором атрибутов.

Для хранения сведений о нескольких версиях стандарта схема базы должна быть расширена за счет дополнительных атрибутов, содержащих темпоральную информацию. Существует несколько подходов к введению таких расширений; в данной работе предлагается использовать темпоральную реляционную модель [8], основанную на реляционной модели, но добавляющую каждой сущности дополнительные атрибуты. При использовании такой модели не обязательно наличие специальной темпоральной СУБД – база данных может обслуживаться и реляционными СУБД, являющимися на сегодняшний день наиболее распространенными.

Двумя обязательными атрибутами, добавляемыми темпоральной моделью, являются начальное и конечное время периода жизни сущности – временного отрезка, на протяжении которого имеет силу конкретное состояние сущности. Областью значений таких атрибутов в нашем случае являются версии стандарта. Отметим, что темпоральные атрибуты добавляются только к сущностям, соответствующим интерфейсам, входящим в стандарт; для данных о дистрибутивах и приложениях они не требуются.

Программа развития инфраструктуры LSB

Одним из стандартов, специфицирующим интерфейсы ОС Linux, которые должны предоставляться приложениям, является Базовый Стандарт (Linux Standard Base, LSB). Разработка стандарта ведется

международным консорциумом Linux Foundation, объединяющем ведущих участников рынка Linux. Основным наполнением стандарта являются списки библиотек, которые должны присутствовать в системе, и списки бинарных символов, которые должны экспортироваться этими библиотеками. Стандарт активно развивается, включая все большее количество элементов – последняя версия (LSB 4.0) специфицирует более 38.000 функций из 57 библиотек, при этом за 4 года, прошедшие с момента выпуска версии 3.0, в стандарт было добавлено более 30.000 функций.

Такое стремительное развитие стандарта выявило ряд проблем в процессе его разработки и используемой для этого инфраструктуре – в частности, отмечалось отсутствие механизма анализа кандидатов на включение в стандарт и трудности в непосредственном использовании текста стандарта, уже в версии 3.0 насчитывавшего несколько тысяч страниц. В 2006 году, в продолжение работ по формализации стандарта LSB [9], стартовала совместная Программа Linux Foundation и Института Системного Программирования РАН (ИСП РАН) по развитию инфраструктуры стандарта LSB. Целью программы являлось разрешение существовавших на тот момент проблем путем создания информационно-аналитической системы, одновременно способствующей быстрому развитию стандарта и обеспечивающей легкость его использования пользователями – разработчиками дистрибутивов и приложений Linux.

К моменту начала Программы в инфраструктуре для разработки стандарта уже присутствовала база данных, содержавшая информацию о стандартизованных элементах. База данных использовалась для генерации частей текста спецификации (списков библиотек, бинарных символов, утилит и т.п.), заголовочных файлов и библиотек-заглушек Среды Разработки LSB (LSB Development Environment – среды, призванной облегчить создание приложений, совместимых со стандартом), а также для создания примитивных тестов, проверяющих наличие различных

элементов (команд, библиотек, бинарных символов) в системе.

В ходе Программы по развитию инфраструктуры LSB в ИСП РАН были выполнены следующие работы:

- разработана схема расширения базы данных для хранения информации о существующих дистрибутивах и приложениях Linux, созданы инструменты для автоматизированного сбора этой информации [10]. В настоящее время в базе содержится информация о 200 дистрибутивах и 1200 приложениях;
- в рамках разработки портала LSB Navigator созданы инструменты для анализа данных о дистрибутивах и приложениях Linux в процессе разработки стандарта LSB с целью выявления кандидатов на стандартизацию, а также проверки ряда формальных требований, предъявляемых к кандидатам на включение в спецификацию;
- разработано темпоральное расширение схемы базы данных, что позволило использовать базу для хранения информации обо всех выпущенных версиях стандарта. В инструменты, работающие с базой, была добавлена возможность создания артефактов, соответствующих любой заданной версии стандарта. Более того, в ряде продуктов, создаваемых на основе базы данных, также имеется поддержка нескольких версий стандарта – так, Среда Разработки LSB может быть использована для создания приложений, соответствующих любой заданной версии LSB.

В настоящее время ведутся работы по обеспечению поддержки профилей в инфраструктуре LSB, что обусловлено потребностью создания профиля стандарта для мобильных устройств.

Текущая схема окружения Базового стандарта Linux приведена на Рисунке 2.

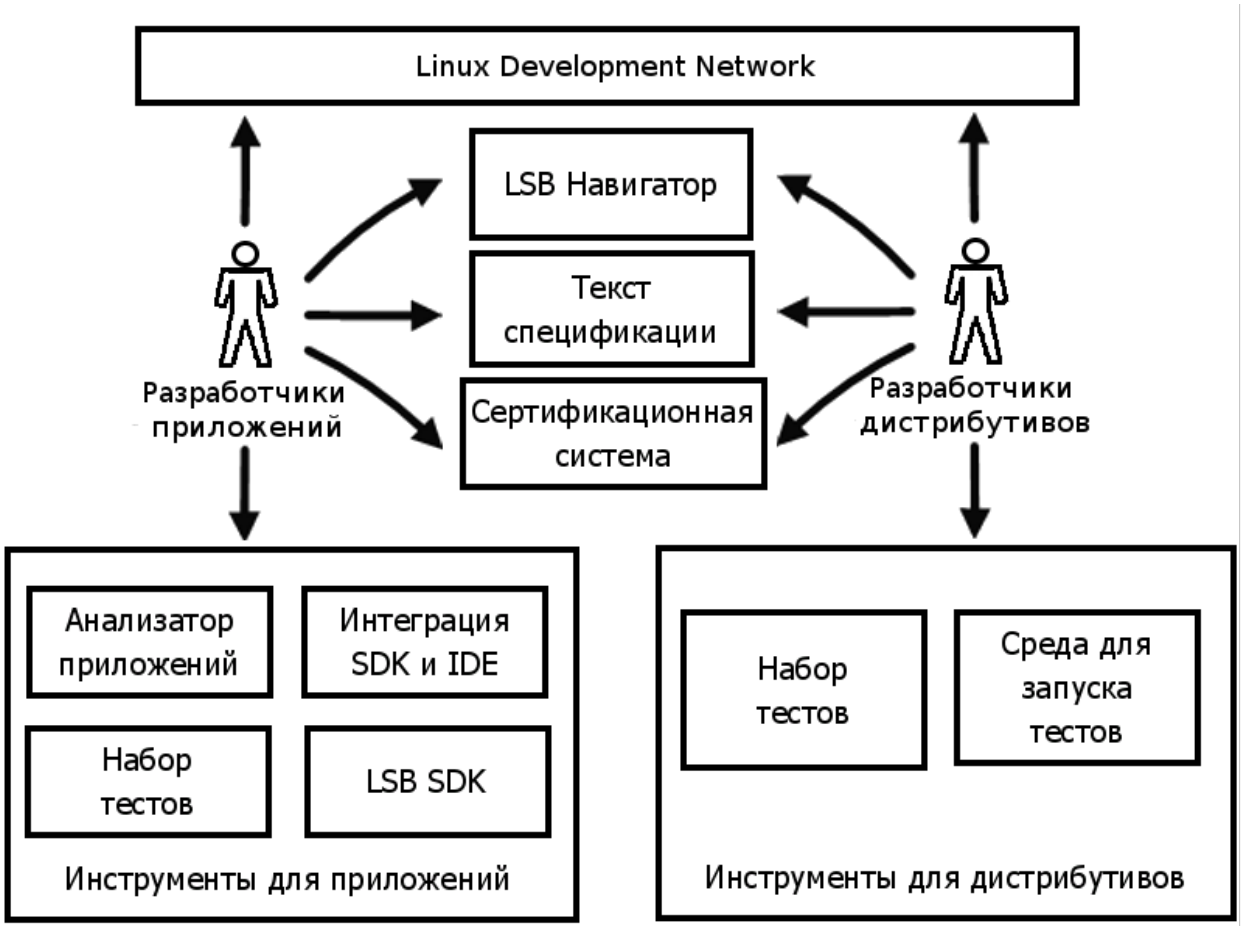

Рисунок 2. Схема инфраструктуры Базового стандарта Linux.

Проект LSB Infrastructure продемонстрировал практическую состоятельность предложенного в данной работе метода развития интерфейсных стандартов ОС Linux. Построенная в ходе проекта инфраструктура Базового стандарта Linux позволила автоматизировать анализ существующих дистрибутивов и приложений Linux, существенно ускорив процесс принятия решений по стандартизации тех или иных интерфейсов. Использование единой базы данных для создания различных сопутствующих стандарту артефактов позволяет обеспечивать согласованность таких артефактов с текстом стандарта и между собой. Кроме того, инфраструктура может быть использована для создания профилей Базового стандарта Linux.

Литература

1. The LWN.net Linux Distribution List. // [HTTP] http://lwn.net/Distributions/

2. E. Raymond. The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. // O'Reilly Media, Inc.; Revised & Expanded edition, 2001.

3. A. Josey. API Standards for Open Systems. // The Open Group, 2001. [HTTP] http://www.opengroup.org/austin/papers/wp-apis.txt

4. System V Application Binary Interface Draft. 24 April, 2001. // [HTTP] http://refspecs.freestandards.org/elf/gabi4+/contents.html

5. Linux Standard Base Core Specification 3.2. Executable And Linking Format (ELF). // [HTTP] http://refspecs.freestandards.org/LSB_3.2.0/LSB-Core-generic/LSB-Core-generic/elf-generic.html

6. M. Tim Jones. Anatomy of Linux dynamic libraries. // IBM developerWorks, 2008. [HTTP]

http://www.ibm.com/developerworks/linux/library/l-dynamic-libraries/

7. Coding practices for compatibility. // Hewlett-Packard Developer & Solution Partner Program, 1999. [HTTP] http://sysdoc.doors.ch/HP/compat.pdf

8. Abdullah Uz Tansel. Temporal Relational Data Model. // IEEE Transactions on Knowledge and Data Engineering. May-June 1997, vol. 9, N3, pp. 464-479.

9. В.В. Кулямин, А.К. Петренко, В.В. Рубанов, А.В. Хорошилов. Формализация интерфейсных стандартов и автоматическое построение тестов соответствия. // Информационные технологии, N8, с.2-7. М., Новые технологии, 2007.