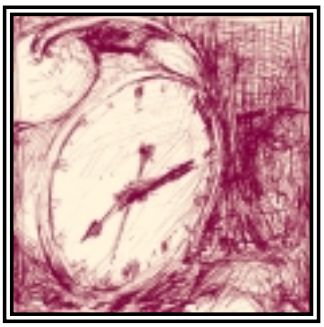


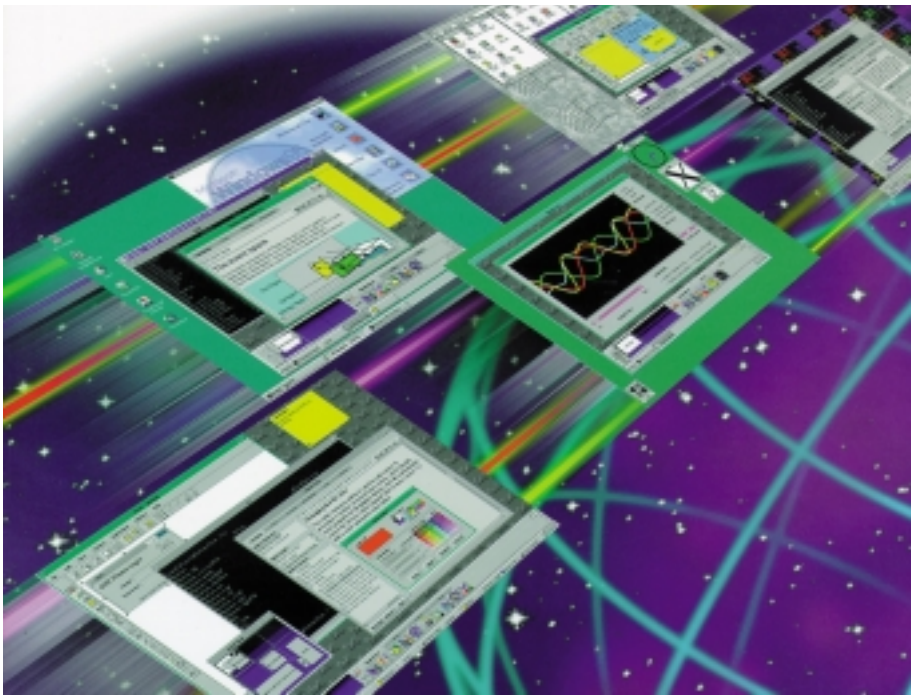
ГРАФИЧЕСКАЯ ОБОЛОЧКА PHOTON — РЕВОЛЮЦИЯ В МИРЕ ИНТЕРФЕЙСОВ

Сергей Ющенко

Графическая среда Photon реализовала многие мечты программистов.

Когда в 1982 году на рынке появилась ОС QNX – первая операционная система на базе микроядра, – она тут же привлекла внимание всех специалистов, от производителей микроконтроллеров до проектировщиков распределенных систем управления, объединив в себе миниатюрность DOS, мощность UNIX и дополнив этот набор распределенной обработкой данных в реальном времени.

Со временем требования к системам растут. И сегодня уже ни один пользователь не представляет себе работу без графического оконного интерфейса. Однако разговор о графической среде наводит разработчиков на мысль о неисчислимых мегабайтах памяти, которые требуются для X Window System или продуктов фирмы Microsoft. И QNX Software Systems Ltd. еще раз показала, что нет ничего невозможного, разработав графическую оконную систему на базе микроядра – Photon.



Думаю, что не найдется программиста, который не мечтал бы о графической среде, которая

- легко умещается в 300 К памяти,
- показывает высокую производительность даже на самой дешевой аппаратной платформе,
- легко масштабируется для использования в разработках самого различного назначения,
- способна отображать свои окна в среде MS Windows или X Window System,
- имеет расширенные средства для проектирования приложений.

Сегодня эти, казалось бы, несбыточные мечты стали реальностью, воплотившись в графическом пакете Photon. Многие возможности, такие как распределенное по сети графическое пространство и низкие требования к аппаратным ресурсам, реализованы за счет возможностей самой операционной системы QNX. Но обо всем по порядку...

Идеология

Photon представляет собой оконную графическую систему, которая по своему подходу к реализации графического интерфейса коренным образом отличается от всех существовавших ранее систем.

Чтобы понять идеологию Photon, вы должны забыть все, что знали ранее о графических оболочках. Пока отвлекитесь от деталей реализации программного обеспечения. Взгляните на графи-

ку не как программист, а как человек, знакомый с оптикой на уровне средней школы. Для начала представьте себе прозрачный параллелепипед. Пространство, которое он ограничивает, в терминах Photon называется пространством событий. Грань параллелепипеда, обращенная к вам, является экраном компьютера (рис. 1). То есть вы находитесь наверху пространства событий. Противоположная грань представляет собой «корневую» плоскость. В этом пространстве размещаются объекты, которыми оперирует Photon, – регионы. Они представляют собой прямоугольные области, расположенные параллельно корневой плоскости, и являются агентами QNX-процессов в пространстве событий. С помощью регионов задача получает сведения о происходящем и с их же помощью доставляет свою информацию. Программа, работающая под управлением Photon, может создавать любое количество регионов и управлять их атрибутами (размером, местоположением в пространстве событий, чувствительностью, прозрачностью и так далее). Механизм передачи информации между регионами реализован на основе сообщений на языке Photon, называемых событиями. События в Photon имеют материальную основу – они представляют собой набор таких же прямоугольных областей, как и регионы, и перемещаются в пространстве событий в двух направлениях: к корневой плоскости или обратно. Регионы могут передавать различные типы событий, такие как события отображения, мыши, клавиатуры и события для позиционирования окон. Если использовать аналогии из области физики, то события являются потоком фотонов, воздействующих на регионы. По отношению к событиям регионы обладают двумя важными свойствами: чувствительностью и прозрачностью. Эти понятия говорят сами за себя. То есть если регион не чувствителен к определенному типу события, он его просто не замечает, если же он не прозрачен для события, проходящего через него, то он будет модифицировать набор прямоугольников события, вырезая области пересечения, в результате чего образуется набор меньших прямоугольников, представляющих собой оставшуюся часть события, продолжающего движение по пространству событий.

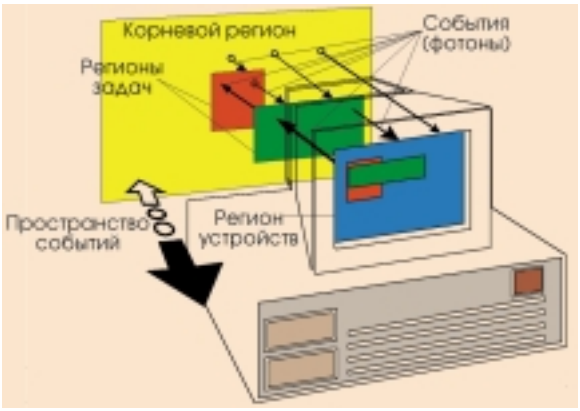


Рис. 1. Графическая метафора системы Photon

Следующий регион, чувствительный к этому событию, будет получать уже новый набор прямоугольников. На рисунке 2 показано событие, полученное регионом Г после прохождения через регионы А, Б и В. Непрозрачность регионов также позволяет избежать нежелательных эффектов прохождения событий. Допустим, что регион Б посылает событие отображения в направлении графического региона (говоря проще, окно Б пытается отобразить себя на экране). Так как регион В, расположенный перед регионом Б, непрозрачен для событий Б, то он вырезает часть события таким образом, чтобы оно не изменило область региона В на экране.

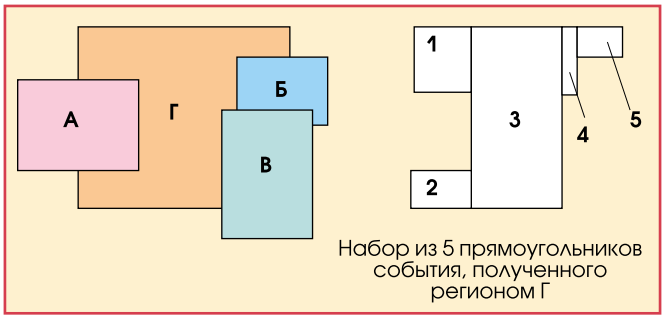


Рис. 2. Прохождение событий через непрозрачные регионы

Регионы драйверов

Подробнее стоит остановиться на регионах драйверов. Сам Photon не знает, как что-либо отображать на экране или воспринимать нажатие клавиши на клавиатуре. Фактически он только обрабатывает события и управляет наборами их прямоугольников.

За отображение информации на экране отвечает графический драйвер, который, как и все драйверы, представляет собой обыкновенную QNX-задачу и использует свой регион для приема графических событий и отображения их на экране, обеспечивая необходимый сервис для остальных задач.

Обработку событий клавиатуры, мыши или сенсорного экрана производит отдельный драйвер, называемый Pointer.

Выполнение драйвера как отдельной задачи имеет ряд преимуществ. Во-первых, это экономит ресурсы вашей системы, так как один драйвер отвечает за те или иные функции, избавляя другие задачи от необходимости решать эту проблему самостоятельно. Во-вторых, взаимодействие с драйвером на основе передачи сообщений позволяет запускать драйверы на отдельном узле сети. На рисунке 3 вы можете видеть, как размещаются регионы драйверов в пространстве событий.

Регион устройств разделяет пространство событий на две части. Регионы драйверов находятся в ближней к пользователю части, регионы остальных программ размещаются между регионом устройств и корневым регионом.

Графический регион находится ближе всех к пользователю. Все события, доходящие до него, он отображает на физическом экране.

Регион обработки мыши принимает информацию от мыши или сенсорного экрана и отправляет ее региону устройств, который, в свою очередь, передает одно событие программам пользователя, а второе графическому региону для отображения на экране перемещения указателя.

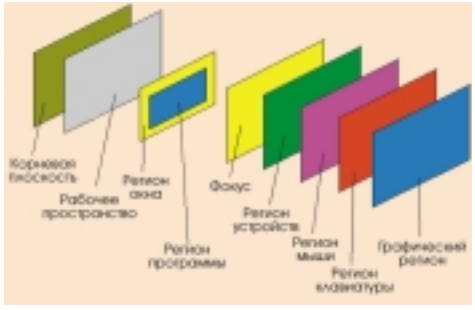


Рис. 3. Размещение регионов драйверов

Окно, распределенное по сети

В Photon графический драйвер может использовать регион, меньший чем высота или ширина пространства событий. Таким образом, вы можете использовать несколько графических драйверов на разных узлах сети для отображения различных частей пространства собы-

тий. То есть пользователь может легко переносить окна с одного физического экрана на другой (рис. 4). Представим ситуацию из будущего отечественных предприятий (и настоящего западных). Предположим, оператор на фабрике имеет переносной компьютер (НРС), снабженный беспроводным сетевым интерфейсом. Оператор может подойти к основному управляющему компьютеру, перенести окно с его экрана на НРС и продолжать следить за процессом, прогуливаясь по фабрике.

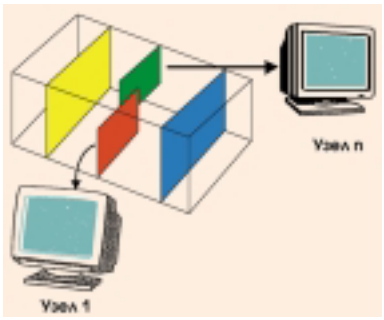


Рис. 4. Перенос окна с одного экрана на другой

Работа с удаленным дисплеем

Предположим, вы хотите посмотреть, что происходит на экране компьютера, расположенного в другом здании (оператор столкнулся с неразрешимой для себя задачей и просит помощи у системного программиста). Для этого необходимо только создать регион, чувствительный к графическим событиям и прозрачный для них, и вставить его в пространство событий нужного вам узла. Так как этот регион прозрачен для графических событий, то он не оказывает влияния на события, происходящие на удаленном узле. Он перехватывает события и перенаправляет их по сети в ваше пространство событий. Такая реализация перехвата событий практически не оказывает влияния на производительность системы в целом.

Администратор оконного интерфейса

Возможность региона фильтровать и воспроизводить события упрощает реализацию администратора оконного интерфейса.

По внешнему виду и функциональным возможностям оконный интерфейс выполнен в стиле известного стандарта OSF Motif [1]. В Photon, как и в любой другой оболочке, пользователь получает возможность управления графическими окнами: перемещения, изменения размера, сворачивания в пиктограмму и так далее.

Когда администратор окон начинает работу, он вставляет свой регион в пространство событий непосредственно перед корневым регионом. Этот новый регион представляет собой рабочее пространство или фон, который пользователь видит на экране. Когда запускается очередная прикладная программа, администратор окон выделяет в ее распоряжение два региона: регион программы, предназначенный для ее использования и принадлежащий только ей, и немного больший по размеру регион окна, о котором программа пользователя может даже не подозревать. В регионе окна менеджер размещает различные атрибуты управления, которые позволяют пользователю перемещать окно, изменять его размер и тому подобное.

Фокусировка

Когда вы работаете с клавиатурой, отображение вводимой информации, как правило, происходит в одном (активном) окне. Вопрос заключается в том, по какому признаку определять активность, или, другими словами, фокусировку. Существуют два способа фокусировки, которые пользователь может выбирать по своему усмотрению: активным считается окно, в пределах которого расположен указатель мыши либо в пределах которого пользователь нажал кнопку мыши. Для Photon фокусировка выражается в том, что регион активного окна становится чувствительным и непрозрачным для событий, исходящих от региона клавиатуры.

Обработка событий

Как и X Window System, Photon использует архитектуру клиент/сервер. Сервером выступает ядро – Photon, обеспечивающее поддержку пространства событий. Любой другой процесс, будь то драйвер или программа пользователя, представляет собой клиентскую часть. Интерфейс между клиентом и сервером базируется на механизме передачи сообщений, используемом в QNX.

Программа, которой принадлежит тот или иной регион, может получать информацию о произошедшем событии одним из трех способов:

- запрос,
- синхронное оповещение,
- асинхронное оповещение.

Используя метод запросов (пример 1), программа производит циклический опрос ядра Photon на предмет прихода того или иного события в удобное для нее время. Для большинства программ этот метод является достаточно неэффективным, но может быть полезным для реализации высокоскоростной анимации, когда программа выполняет опрос в промежутках между выполнением графических или других функций.

Большинство программ использует метод синхронного оповещения (пример 2). Программа посылает запрос ядру Photon, который отвечает только тогда, когда произошло событие, в котором программа заинтересована. Все время с момента посылки запроса до получения ответа на него Photon блокирует программу, то есть она находится в состоянии ожидания и не производит никаких действий.

Если вам нужно больше гибкости, например, когда вы пишете программу-сервер, которая должна принимать и обрабатывать сообщения, приходящие от других задач, вы должны использовать метод асинхронного оповещения (пример 3). В этом случае программа вызывает функцию, устанавливающую заместитель (proxy) [3], [8], которую Photon будет «включать» после прихода события. Программа, выполняющая свои основные функции как сервер, будет получать среди сообщений от кли-

Пример 1

```
/* Размер буфера события */
#define EVENT_SIZE  sizeof( PhEvent_t) + 1000

void main( int argc; char *argv[])
 {
  PhEvent_t *event;  /* Буфер событий */
  ........

  while( 1) {
   ............ /* Код пользователя */

   /* Запрос к Photon с моментальным ответом */
   switch( PhEventPeek( event, EVENT_SIZE)) {
     case Ph_EVENT_MSG:   /* Обработка события */
               PtEventHandler( event);
               break;
     case -1:             /* Ошибка запроса */
               perror( "PhEventPeek failed");
     case  0:             /* Нет доступных событий */
               break;
     }
   }
 }
```

Пример 2

```
/* Размер буфера события */
#define EVENT_SIZE  sizeof( PhEvent_t) + 1000

void main( int argc; char *argv[])
 {
  PhEvent_t *event;  /* Буфер событий */
  ........

  while( 1) {
   ............ /* Код пользователя */

   /* Запрос к Photon с ожиданием ответа  */
   switch( PhEventNext( event, EVENT_SIZE)) {
     case Ph_EVENT_MSG:   /* Обработка события */
               PtEventHandler( event);
               break;
     case -1:             /* Ошибка запроса */
               perror( "PhEventNext failed");
               break;
     }
   }
 }
```

ентов извещения от заместителя о приходе события.

Разработка программ

Наверное, не ошибусь, если скажу, что эта глава вызовет наибольший интерес у программистов. В среде Photon могут создавать программы как те, кто поработал с X или MS Windows, так и люди, впервые столкнувшиеся с графическим интерфейсом. Разработчики Photon позаботились о простоте и мощности интерфейса программиста, снабдив его редактором, обеспечивающим визуальное программирование, – Application Builder.

Программы Photon могут использовать несколько уровней программного обеспечения, представленного библиотечными функциями. На нижнем уровне пользователь может работать с графическими примитивами типа «линия», «текст» и им подобными. На верхнем уровне программирования пользователю доступны готовые элементы интерфейса, называемые виджеты (widgets) [1], [2]. Виджеты представляют собой объекты, поведением которых на экране может управлять Photon. Преимущества такого подхода очевидны: программист не должен обладать информацией о деталях реализации того или иного объекта, ему нужно знать только функции, организующие интерфейс с этим виджетом, такие как создание, уничтожение и управление ресурсами. Наряду с геометрическими ресурсами, такими как цвет, размер, положение в пространстве событий, каждый виджет поддерживает ресурс, представляющий собой список функций, определяемых программистом (callbacks) и выполняемых после получения какого-либо события. Например в список ресурсов объекта «кнопка» можно внести функцию открытия нового окна и изменения вида кнопки после ее нажатия.

Виджеты, как правило, не являются уникальными, а представляют собой семейства, образующие класс. Все виджеты, принадлежащие к одному классу, обладают одинаковыми свойствами и могут наследовать свойства своих базовых классов. Помимо виджетов, представленных в библиотеках Photon, программист может использовать свои собственные виджеты или перенесенные из среды X Window System.

Листинги программ управления ресурсами виджетов на примере знаменитого «Hello World» вы можете запросить по e-mail. Пример использования возможностей Application Builder для реализации той же самой программы дан на рис. 5.

Пример 3

```
/* Размер буфера события */
#define EVENT_SIZE  sizeof( PhEvent_t) + 1000

void main( int argc; char *argv[])
 {
  pid_t     pid;       /* Идентификатор процесса,
                          посылающего сообщение */
  struct my_data msg;  /* Структура данных пользователя*/
  PhEvent_t *event;    /* Буфер событий */
  ........

  PhEventArm(); /* Установка proxy */

  while( 1) {
   pid = Receive( 0, &msg, sizeof( msg)); /* Прием сообщений */

   /* Запрос к Photon на предмет принадлежности сообщения
      и считывание события при положительном ответе      */
   switch( PhEventRead( pid, event, EVENT_SIZE)) {
     case Ph_EVENT_MSG:   /* Обработка события */
               PtEventHandler( event);
               break;
     case  0:             /* Сообщение не от Photon */
               ............ /* Код пользователя */
               break;
     case -1:             /* Ошибка запроса */
               perror( "PhEventRead failed");
               break;
     }
   }
 }
```

Несколько практических аспектов

Компактность

Самими разработчиками Photon рекламируется как графическая оболочка для встроенных систем (embedded systems) [5], [6], [7]. Скорее всего, само понятие «встроенные системы» требует некоторой конкретизации. Речь идет о любой реализации компьютера, отличной от той, которую вы видите на своем столе. Под эту классификацию попадают, в основном, компьютеры с ограниченными ресурсами памяти, как дисковой, так и оперативной.

Очевидно, что установка 16-32 Мбайт ОЗУ на промышленный контроллер только для возможности визуализации процесса в X Window System была бы непозволительной роскошью. Но на добавление 1-2 Мбайт ради возможности получить удобный графический интерфейс, вероятнее всего, согласятся многие.

Так, например, необходимый для работы пользовательской программы набор, включающий ядро Photon, графический драйвер и администратор окон, занимает не более 360 К ОЗУ.

Масштабируемость

Модульная архитектура Photon позволяет настраивать систему по вашему требованию. Например, когда вы не особенно ограничены в аппаратных ресурсах, то есть работаете на компьютере с 4-8 Мбайт ОЗУ, то легко можете себе позволить использовать менеджер рабочего стола (Photon Desktop Manager) либо свои сервисные утилиты. А программу, разработанную для мощ-

ной аппаратной платформы, можно использовать в среде с ограниченными ресурсами без изменения кода, отказавшись от тех или иных возможностей.

Быстродействие

Photon действительно может снизить стоимость вашего проекта. Он достигает скорости воспроизведения графических объектов на экране, сравнимой с быстродействием графического драйвера. Таким образом, вы можете в системе с 386-м процессором и простой VGA-картой работать в оконной среде примерно с таким же уровнем комфорта, как на 486DX2 в среде MS Windows.

Взаимодействие с другими графическими средами

Как поступить в ситуации, если пользователь работает в другой графической среде? Вы можете отображать окна, созданные в Photon в таких оболочках, как X Window System и MS Windows. Для этого просто запускается дополнительная утилита, перенаправляющая события Photon в другую среду.●

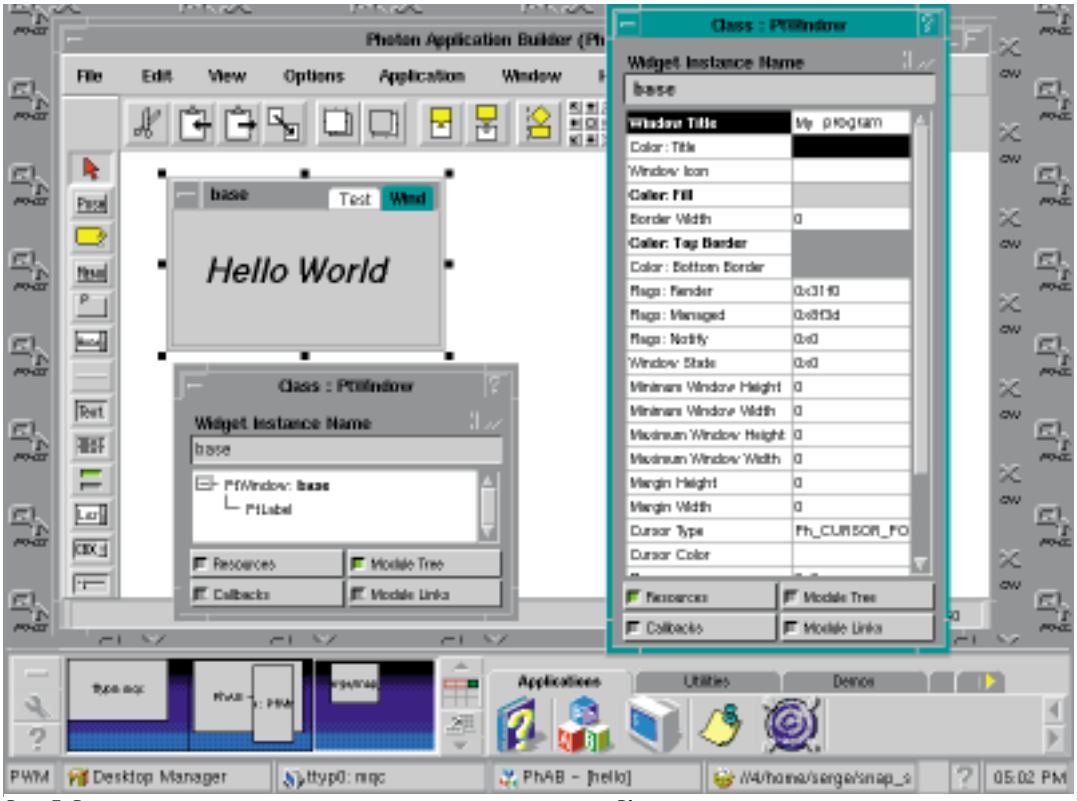


Рис. 5. Возможности визуального программирования в системе Photon

Рекомендуемая литература

1. Valerie Quercia and Tim O'Reilly. X Window System User's Guide/Vol. 3. — Canada: QNX Software Systems Ltd. , 1993.
2. Photon microGUI. Programer's Guide.- Canada: QNX Software Systems Ltd., 1995.
3. Макарьев К., Кондратюк А. Знакомство с WATCOM C для QNX// Монитор. – 1994. — № 5. — С. 58— 66.
4. Dan Hildebrand. Building GUI Applications for QNX// QNXnews.- 1993. — Vol. 7. — № 4. — P. 30-31.
5. Dan Hildebrand. Project Photon — Embedded Windowing for QNX //QNXnews. — 1993. — Vol. 7. — № 4. — P. 17-22.
6. Rob Oakley. QNX Microkernel Technology: A Scalable Approach to Realtime Distributed and Embedded Systems. – Canada: QNX Software Systems Ltd. , 1994.
7. It's Small, It's Scalable, and It Connects Seamlessly to Desktop GUIs — It's the Photon microGUI!// QNXnews. — 1995. — Vol. 9. — № 2. — P. 7-10.
8. QNX System Architecture. — Canada: QNX Software Systems Ltd., 1995.