УДК 519.683.8

А.Б.Веретенников, канд. физ.-мат. наук (Уральский федеральный университет им. первого Президента России Б.Н. Ельцина, Екатеринбург);
e-mail: alexander@veretennikov.ru

СОЗДАНИЕ МНОГОПАНЕЛЬНЫХ ИНТЕРФЕЙСОВ ПРОГРАММ НА СКРИПТОВЫХ ЯЗЫКАХ

Рассмотрены способ и преимущества создания оконных интерфейсов любого уровня сложности на скриптовых языках. В качестве примера проанализирована разработка многопанельного интерфейса. Модель, не зависящая от конкретной реализации, представлена в виде программной библиотеки.

Ключевые слова: многопанельный интерфейс; JavaScript GUI; скриптовые языки.

A. B.Veretennikov (Ural Federal University named after the first President of Russia B.N.Yeltsin, Yekaterinburg)

CREATING DOCKABLE WINDOWS INTERFACES OF PROGRAMS IN SCRIPTING LANGUAGES

Scripting languages such as JavaScript are proven to be the effective way to build Web interfaces, but have limited features to build interfaces of Desktop applications. But a method to create complex user interfaces of the Desktop applications exists. For example we are considering the dockable windows interface. A user interface usually consists of some panels. The docking framework provides a way to change layout. The user can move panels by using drag and drop from one place to another and can change size of the panels. Dockable windows are used in the Microsoft Visual Studio and Eclipse. In this paper a docking framework for scripting languages, its model and API are presented. The goal is to develop as simple and concise API as possible. An implementation of the model is provided. Model itself is not dependent on the implementation. Building user interface using scripting languages have advantages described in the paper. Scripting languages are often referred to as glue languages or ever system integration languages. They are intended for connecting components developed with other languages. With scripting languages the system business logic can be isolated from the user interface. The user interface can be rapidly developed, supported and updated.

Keywords: Docking framework; JavaScript GUI; Scripting languages.

Введение

Скриптовые языки активно развиваются. Широко используют JavaScript, новый стандарт которого [1] выпущен в июне 2015 г. Реализация этого языка от Microsoft входит в состав операционной системы Windows. Google ведет разработку V8, его альтернативной реализации [2].

Основная область применения JavaScript это клиентское Web-программирование, но этот язык – язык общего назначения, применяют,

например, в *WScript* (*Windows Script Host*) от Microsoft и в Node.js, основанном на Google V8.

Часто скриптовые языки не имеют полных возможностей для создания интерфейсов пользователя Desktop-приложений. Для VBScript или JavaScript используют средства браузера.

Интерфейсы, созданные с помощью данных средств не похожи на интерфейсы обычных программ. Для их создания в браузере активно развивают библиотеки Sencha Ext Js и AngularJs от Google, которые сложны в разработке и предназначены в основном для дизайна Web-сайтов. Также применяют *HTA* (*HTML* (*HyperText Markup Language) Application*) [4], использующие возможности браузера.

Широко известны такие языки как Perl [3], Python и Ruby. Интерфейсы пользователя для них можно создавать, например, с помощью подключения GTK+, однако подобные средства обычно приходят из Linux и не используют все возможности платформы Windows, трудны в установке и интерфейсы, созданные с их помощью, часто смотрятся чужими в Windows.

Библиотека *WSO* (*WindowSystemObject*), разработанная автором, предназначена для быстрого создания интерфейсов разного уровня сложности [5, 6]. При этом интерфейс программы состоит из обычных оконных элементов управления или контролов Windows и программа, соответственно, выглядит естественным образом. Далее мы называем оконный элемент управления (контрол) – компонентом.

Разработан визуальный редактор форм. WSO реализована с помощью технологий объектная модель компонентов — *COM* (*Component Object Model*) и автоматизация [7], которые разработаны Microsoft для создания внешних объектов, которые можно использовать в скриптах. Несмотря на то, что активно развивается технология .NET, для Windows технология COM остается одним из основных вариантов для создания объектов или библиотек на языках C и C++. Отметим, что существует XPCOM, аналог COM, созданный Mozilla.

Можно также обратить внимание на KiXForms, позволяющий создавать интерфейсы пользователя в скриптах, используя COM, но последняя версия выпущена в конце 2007 г., не поддерживает последующие версии Windows и есть проблемы с обработкой событий.

Скрипты и «модель-представление-контроллер»

Шаблон проектирования *MVC* (*Model View Controller*) [8] предполагает разделение системы на три части:

- *модель* (*model*) − данные системы, например, в реляционной или XML базе данных, и методы их обработки;
- *представление*, *вид* (*view*) − интерфейс пользователя;
- *контроллер* (*controller*) − связывает представление с моделью.

Данный шаблон активно применяют при создании Web-приложений (эти части легко выделить), однако для создания Desktop-приложений его применяют реже. MVC показывает важность отдельной разработки представления данных. Можно разрабатывать интерфейс пользователя на скриптовом языке, а саму бизнес логику на другом языке программирования, например, C или C++.

Скриптовые языки называют языками системной интеграции [9, 10]. С их помощью объединяют сущности в единую систему. Эти цели ставили при разработке скриптового языка Tcl/Tk [11] и там, в отличие от JavaScript графическую библиотеку для Desktop- приложений [12] создавали изначально. Если, например, есть интерфейс, написанный на скриптовом языке, и бизнес логика заключена в отдельных COM-компонентах, то мы достигаем:

1) Максимальной изоляции бизнес логики системы от интерфейса пользователя, что облегчает ее переиспользование. Бизнес логику системы пользователи могут использовать в своих скриптах независимо от существующего интерфейса системы.
2) Быстрого изменения интерфейса пользователя без перекомпиляции бизнес логики системы, что значительно ускоряет разработку интерфейса систе-

мы и облегчает ее обновление.

3) Максимальной производительности, которую обеспечивает использование не скриптового языка для разработки бизнес логики.
4) Более простого и быстрого способа создания сложных видов интерфейсов, например, многопанельных, так как задача создания интерфейса изолирована от остальных задач.

Многопанельный интерфейс

Многопанельный интерфейс, позволяет пользователю выполнять несколько задач параллельно более чем на 10% быстрее по сравнению с традиционными интерфейсами, а некоторые действия по работе с окнами – на 20% [13, 14], и он оптимален для решения проблемы ограниченного места на экране [15]. На рис. 1 представлен многопанельный интерфейс (WSO), одна из панелей которого (Doc1) перемещается пользователем на новое место.

Существуют библиотеки, в том числе, коммерческие, для создания подобных интерфейсов, например, JIDE Docking Framework, Docking Frames, InfoNode Docking Windows, Telerik Dock, но они, в основном, созданы для Java и .NET и их нельзя использовать в скриптовых языках.

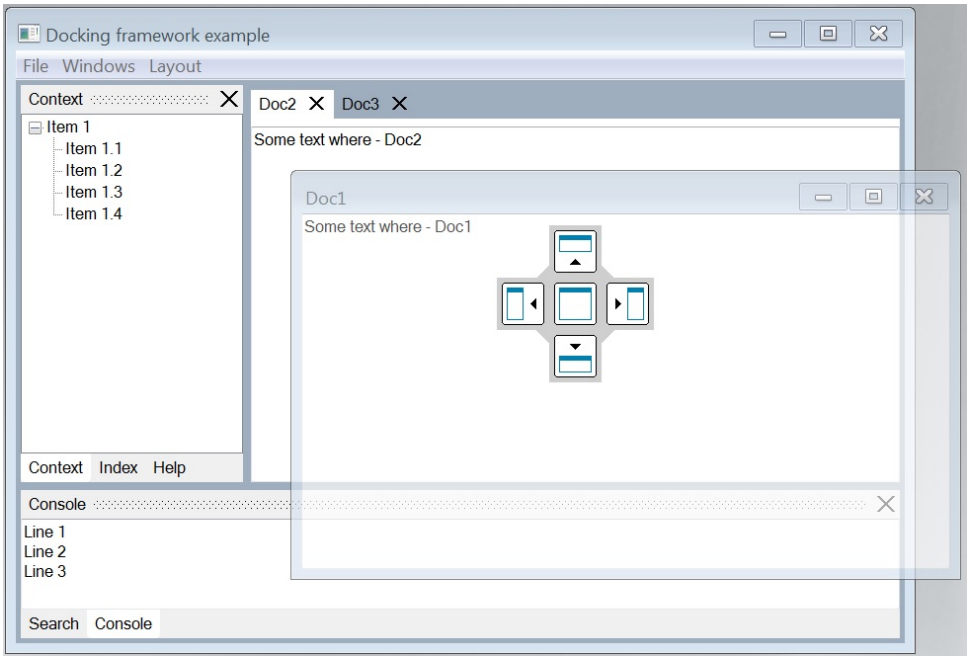

Рис 1. Многопанельный интерфейс, одна из панелей (*Doc1*) перемещается пользователем на новое место.

Для реализации многопанельного интерфейса WSO имеет:

1) *Frame*, одностраничный контейнер для других компонентов. В простейшем случае представляет собой окно прямоугольной формы, без каких-либо вспомогательных элементов, содержащее другие компоненты.
2) *Form* – окно верхнего уровня, имеет рамку.
3) *TabControl*, многостраничный контейнер для других компонентов. Содержит набор одностраничных контейнеров-страниц, каждый из которых имеет закладку. Одновременно видна одна страница. Смена текущей видимой страницы происходит при нажатии на закладку.
4) *PageControl*, многостраничный контейнер, так же как и TabControl, но с другим внешним видом, в котором отдельно от закладок отображен заголовок текущей видимой страницы.

Далее опишем интерфейс прикладного программирования *API* (*Application Programming Interface)*, многопанельного интерфейса. API состоит из набора объектов, каждый из которых имеет методы и свойства. При задании некоторых свойств могут использовать символические константы, такие как AL_LEFT, смысл которых будет прояснен при их использовании. Некоторые свойства – логические параметры, то есть могут принимать значения "TRUE" или "FALSE", соответственно, "Истина" или "Ложь".

Основной компонент – панель для организации многопанельного интерфейса Frame. Пользователь может перетаскивать панели между указанными четырьмя контейнерами. При помещении панели *P* в многостраничный контейнер, т. е. на одну из его страниц, для *P* создается отдельная страница-закладка. При помещении панели *P* в одностраничный контейнер *C*, Frame или Form, доступны два варианта:

1) *P* помещают как дочерний компонент в *C*. В этом случае *P* может располагаться в *C* слева, справа, сверху, снизу или занимать все доступное пространство, т. е. получать значение свойства Align AL_LEFT, AL_RIGHT, AL_TOP, AL_BOTTOM, AL_CLIENT, соответственно.
2) *C* – Frame*: P* помещают в *C* как рядом располагающийся компонент. В

этом случае для *P* и *C* создают общий родительский компонент. Если *P* помещают относительно *C* слева, справа, сверху, снизу (AL_LEFT, AL_RIGHT, AL_TOP, AL_BOTTOM, соответственно), то в качестве родительского компонента создают Frame. При этом дочерние компоненты, *P* и *C*, автоматически разделяют разграничителем с помощью которого можно менять их размеры. Если *P* помещают на *C* (аналогично AL_CLIENT), то в качестве общего родительского компонента создают многостраничный компонент TabControl или PageControl, *P* и *C* помещают на его закладки.

Свойства AllowDockAsChild и AllowDockAsNeighbour определяют для целевого компонента *C*, может ли пользователь поместить *P* в *C* первым или вторым вариантом, соответственно.

Свойства AlwaysDockTab и AlwaysDockPage

Если AlwaysDockTab = "TRUE", то для панели Frame автоматически создается родительский TabControl. Рассмотрим следующий фрагмент кода (JScript): создается Frame с одним дочерним элементом. Если в коде закомментировать или убрать изменение свойства AlwaysDockTab = "TRUE", то будет присутствовать только сам Frame, значение свойства Text не отображается (рис. 2, *а*). Если же в коде присутствует изменение свойства AlwaysDockTab = "TRUE", то будет автоматически создан TabControl, (рис. 2, *б*) значение свойства Text используют в качестве названия закладки.

```
wso = new ActiveXObject("Scripting.WindowSystemObject")
wso.enableVisualStyles = true
form = wso.createForm(10, 10, 210, 140)
form.text = "WSO"
frame = form.createFrame(10, 10, 180, 80)
frame.bevelOuter = wso.translate("BS_RAISED")
frame.textOut(10, 10, "Test")
frame.text = "Frame"
frame.docking.alwaysDockTab = true
```

form.show()

wso.run()

Для перетаскивания компонента пользователь нажимает левую кнопку мышки на заголовок закладки (рис. 2, *б*, затем рис. 2, *в*) и, удерживая кнопку мышки нажатой, перемещает ее. Панель Frame извлекают из закладки и помещают в создаваемую форму Form. TabControl удаляют, если в нем больше нет закладок.

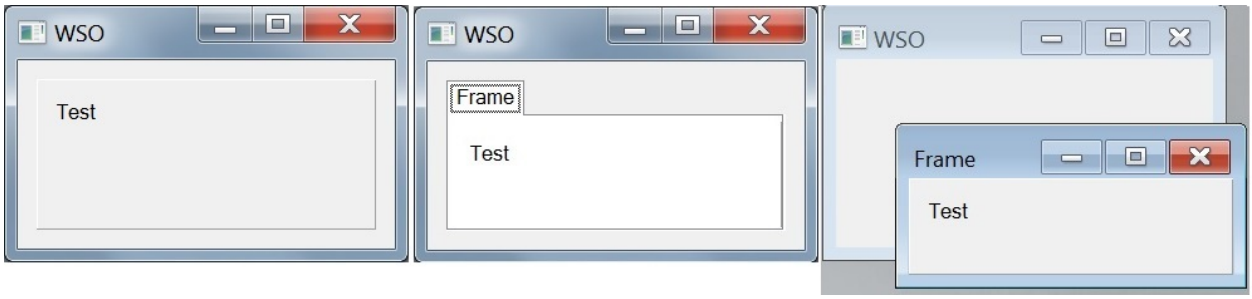

Рис 2. Изменение свойства *AlwaysDockTab* и извлечение панели из родительского контейнера.

Создаваемые автоматически контейнеры отмечают AutoCreated = "TRUE" и могут удалять их автоматически, при извлечении из них дочерних компонентов.

Родительский TabControl создается автоматически при изменении свойства AlwaysDockTab и при перемещении панели в другой компонент, если:

– целевой компонент не многостраничный,

– целевой компонент не автоматически созданная форма, например, при перемещении панели в другую панель Frame.

Если AlwaysDockPage = "TRUE", то для Frame автоматически создается родительский PageControl. Работа с PageControl осуществляется аналогично TabControl, т. е. используют заголовок закладки. Свойства AlwaysDockTab и AlwaysDockPage определяют предпочтительный тип родительского многостраничного контейнера для текущего компонента. Введем понятие *тип многостраничного контейнера* (*ТМК*). Если компонент лежит в многостраничном контейнере, то тип этого контейнера определяет ТМК для компонента. Иначе – ТМК определяется свойствами AlwaysDockTab и AlwaysDockPage.

Перемещение формы с компонентом

Когда перемещаемая форма оказывается поверх целевого компонента, отображаются кнопки, которые определяют, каким образом компонент будет расположен относительно целевого компонента. Кнопки отображаются, только если целевой компонент отмечен DropTarget = "TRUE". Указатель мышки помещают на одну из появляющихся кнопок. Когда пользователь отпустит клавишу мышки, перетаскиваемый компонент будет перемещен на целевой компонент. На рис. 3 представлены зависимости разных сочетаний кнопок от разных условий

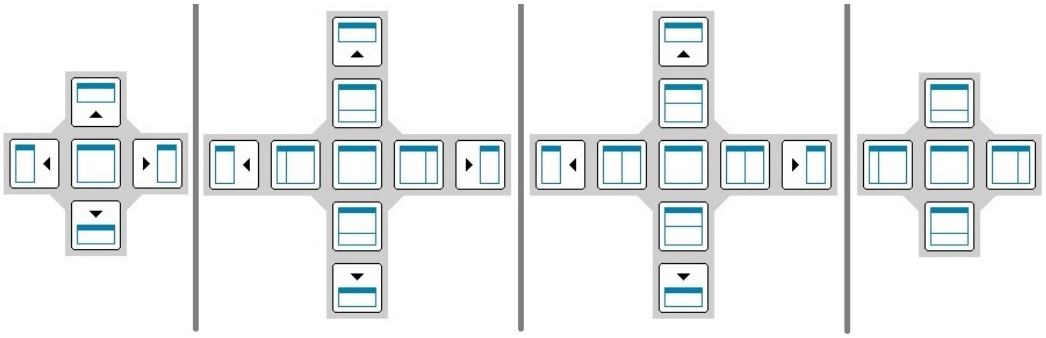

Рис 3. Зависимость разных сочетаний кнопок от разных условий

Рассмотрим 4 примера.

Пример 1 (рис 3, *а*)**:**

– целевой компонент Frame имеет свойства AllowDockAsChild = "FALSE", AllowDockAsNeighbour = "TRUE" и лежит в многостраничном контейнере (TabControl);

– ТМК перемещаемого компонента и целевого контейнера совпадают.

При выборе кнопки в центре перемещаемый и целевой компоненты будут помещены в качестве закладок на многостраничном контейнере. При выборе кнопок по краям перемещаемый компонент будет помещен рядом с целевым компонентом.

Пример 2 (рис 3, *б*)**:**

– целевой компонент *Frame* имеет свойства

AllowDockAsChild = "TRUE", AllowDockAsNeighbour = "TRUE" и лежит в многостраничном контейнере (TabControl).

– ТМК перемещаемого компонента и целевого контейнера совпадают.

Добавляются дополнительные кнопки, которые соответствуют помещению перемещаемого компонента как дочернего компонента в целевой компонент.

Пример 3 (рис 3, *в*)**:**

– Целевой компонент Frame имеет свойства

AllowDockAsChild = "FALSE", AllowDockAsNeighbour = "TRUE".

и лежит в многостраничном контейнере (TabControl или PageControl),

–ТМК перемещаемого компонента и целевого компонента не совпадают, например, перемещаемый компонент имеет AlwaysDockTab = "TRUE", а целевой компонент лежит в PageControl.

Добавляются дополнительные кнопки означающие, что перемещаемый компонент будет помещен рядом с целевым компонентом, но его ТМК будет преобразован в ТМК целевого компонента, т. е. будет создан такой же многостраничный контейнер, как и у целевого компонента.

Пример 4 (рис 3, *г*)**:**

– целевой компонент Frame имеет свойства

AllowDockAsChild = "TRUE", AllowDockAsNeighbour = "FALSE" и

лежит в многостраничном контейнере;

– ТМК исходного компонента и целевого контейнера совпадают, например, исходный и целевой компоненты лежат в TabControl.

Доступны только кнопки для помещения компонента как дочернего в целевой компонент.

Если целевой компонент имеет AllowDockAsChild = "TRUE", то при нажатии на центральную кнопку перемещаемый компонент будет помещен внутрь целевого компонента, с Align = AL_CLIENT. В ином случае при нажатии на центральную кнопку перемещаемый компонент будет помещен как соседний компонент на целевой компонент, т. е. оба они будут помещены на закладки общего многостраничного контейнера.

В дополнение к AlwaysDockPage и AlwaysDockTab свойство PageControlTag позволяет ввести разные виды многостраничных контейнеров одинакового типа, например PageControl с разной раскраской или с разным распо-

ложением закладок, внизу или вверху. Реализованы возможности сохранения и загрузки расположения панелей, создания полупрозрачных и фигурных окон, настройки Task Bar, Tray Icon. Окно может становиться полупрозрачным при его перемещении, чтобы пользователь мог видеть скрытые им части интерфейса при определении его нового расположения.

Модель системы с многопанельным интерфейсом

Мы можем определить последовательность сущностей:

Задача -> Бизнес модуль -> Одностраничный контейнер.

Пара: задача, бизнес модуль – это модель. Бизнес модуль решает задачу и реализован на языке системного уровня. Одностраничный контейнер – соответствующее представление. Контроллер – логика на скриптовом языке, которая осуществляет обработку действий пользователя при работе с представлением. Таким образом, базовый элемент модели реализует MVC. Модель системы с многопанельным интерфейсом представлена на рис. 4.

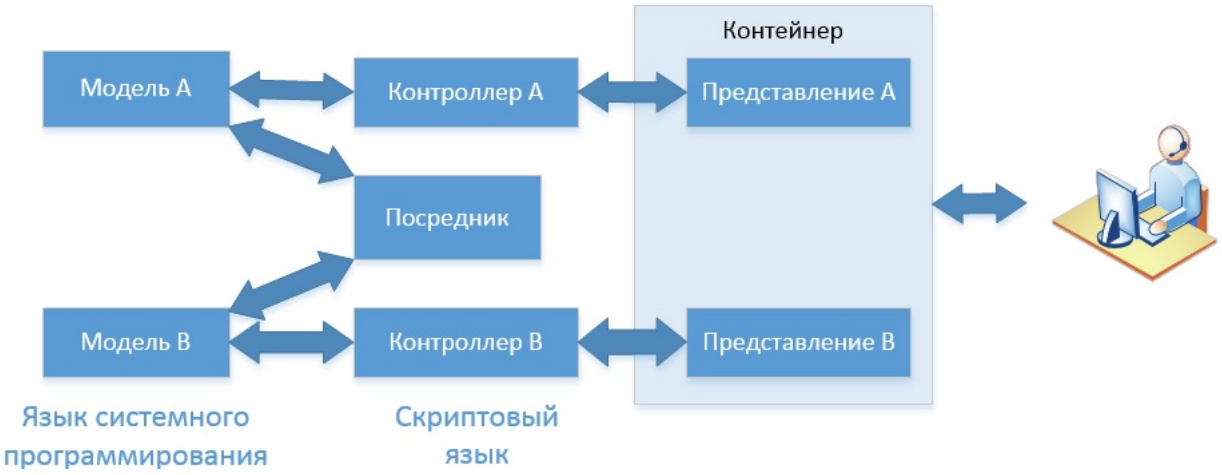

Рис 4. Модель системы с многопанельным интерфейсом.

Последовательности могут объединяться, для этого на уровне интерфейса применяют, например, многостраничный контейнер объединяющий два представления в одно. При этом не возникает нового контроллера. Применение скриптовых языков позволяет изолировать одну задачу от другой. Бизнес модули могут быть связаны между собой, для этого можно создать отдельную сущность – посредник.

Заключение

Рассмотрена разработка многопанельного интерфейса (Docking

Framework), состоящего из панелей, с использованием скриптовых языков. Пользователь может менять расположение панелей с помощью мышки, перетаскивая их из одного оконного компонента в другой, извлекать из основного окна в новые вспомогательные окна, создаваемые при этом автоматически. Данный вид интерфейса пользователя предназначен для решения проблемы ограниченного места на экране. Реализация которого на скриптовом языке позволяет ускорить разработку интерфейса и его последующие поддержку и обновление.

Рассмотрена модель создания многопанельного интерфейса. Данную модель и API можно рассматривать без привязки к конкретной реализации и, таким образом, они могут быть полезны при разработке многопанельных интерфейсов как на скриптовых языках, так и на других языках программирования.

Библиографический список

1) **ECMA-262** [Электронный ресурс] // ECMAScript Language Specification 6th edition : офиц. сайт. URL: http://www.ecma-international.org/publications/standards/Ecma-262.htm/ (дата обращения 01.08.2015).
2) **Google** [Электронный ресурс] // V8 JavaScript engine: офиц. сайт. URL: http://code.google.com/apis/v8/ (дата обращения 01.08.2015).
3) **Wall L., Schwartz R**. Programming Perl. USA: O'Reilly and Associates, 1990. 482 p.
4) **Extreme Makeover: Wrap Your Scripts Up in a GUI Interface** [Электронный ресурс]. URL: https://technet.microsoft.com/en-us/library/ee692768.aspx (дата обращения 01.08.2015).
5) **Веретенников А. Б**. Библиотека для создания оконных интерфейсов на любых скриптовых языках в операционной системе Windows // Информационно-математические технологии в экономике, технике и образовании: тез. докл. Екатеринбург: УГТУ-УПИ, 2008. С. 220 – 221.

6) **WindowSystemObject** [Электронный ресурс]: офиц. сайт. URL: http://veretennikov.org (дата обращения 01.08.2015).

7) **Automation** (formerly known as OLE Automation) [Электронный ресурс]: офиц. сайт. URL: https://msdn.microsoft.com/en-us/library/dt80be78.aspx (дата обращения 01.08.2015).

8) **Krasner G., Pope S.** A cookbook for using the model-view controller user interface paradigm in Smalltalk-80 // Journal of Object-Oriented Programming archive. 1988. Vol. 1, № 3, P. 26 – 49.

9) **Ousterhout J.** Scripting: Higher-Level Programming for the 21st Century // IEEE Computer. 1998. P. 23 – 30.

10) **Schneider J., Nierstrasz O**. Components, Scripts and Glue // Software Architectures – Advances and Applications. Springer. 1999. P. 13 – 25.

11) **Ousterhout J.** Tcl and the Tk Toolkit. USA: Addison-Wesley, 1995. 142 p.

12) **Ousterhout J.** An X11 Toolkit Based on the Tcl Language // Proceedings of the 1991 Winter USENIX Conference, January 1991. P. 105 – 115.

13) **Shibata H., Omura K.** Docking window framework: Supporting multitasking by docking windows // Proc. of APCHI'12. NY: ACM, 2012. P. 227 – 236.

14) **Shibata H., Omura K.** Reducing the cost of window operations by docking windows // International Journal of Innovative Computing, Information and Control. 2013. № 9 (12). P. 4665 – 4679.

15) **JIDE Docking Framework** [Электронный ресурс]: офиц. сайт. URL: http://www.jidesoft.com/products/dock.htm (дата обращения 01.08.2015).