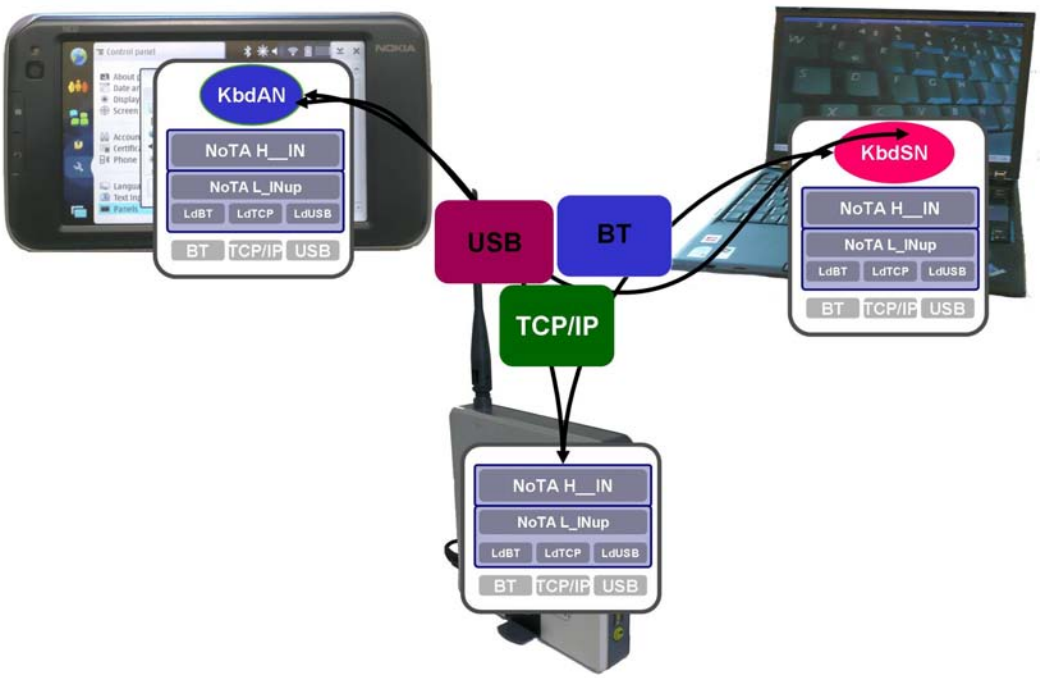

Developing Keyboard Service for NoTA

Kari Keinänen, Jarkko Leino & Jani Suomalainen

ISBN 978-951-38-7168-0 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 7001

VTT, Bergsmansvägen 5, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 7001

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 7001

VTT, Vuorimiehentie 3, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 7028

VTT, Bergsmansvägen 3, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 7028

VTT Technical Research Centre of Finland, Vuorimiehentie 3, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 7028

Series title, number and
report code of publication

VTT Working Papers 107
VTT-WORK-107

Author(s)
Keinänen, Kari, Leino, Jarkko & Suomalainen, Jani

Title
Developing Keyboard Service for NoTA

Abstract
Developed Keyboard Service enables different kinds of keyboards (remote or devices' own) to provide input for users' applications. It is built on top of NoTA-protocol stack, a new service-oriented middleware architecture making service and application development independent of underlying physical transport layers. Three transport layers, TCP/IP, Bluetooth and USB were used and tested with Keyboard Service.

ISBN
978-951-38-7168-0 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN		Project number
VTT Working Papers 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)		28764
Date October 2008	Language English	Pages 17 p. + app. 2 p.

Name of project
DIEM (Devices and Interoperability Ecosystems)

Keywords	Publisher
NoTA, service development, keyboard	VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4520 Fax +358 20 722 4374

Preface

Network on Terminal Architecture (NoTA) is a new middleware technology that has a potential to reshape the development landscape of mobile and embedded services. We here in VTT Technical Research Centre of Finland noted the promise of this technology couple years ago and since then have participated in several NoTA development projects. External keyboard was the first NoTA service we developed. It was presented publicly in the First NoTA conference in June 2008.

The purpose of this publication is to provide a guide for keyboard service users and developers. Also, we want to share some experiences that we have encountered when developing NoTA services.

NoTA is originated by Nokia Research Center. We want to thank persons from Nokia and VTT, who have participated to the development of NoTA and cooperated with us.

Project web pages, containing additional documentation and source code (licensed under GPLv2), can be found from the URL:

http://www.notaworld.org/project/kbd

We warmly welcome all new developers to join NoTA keyboard service project or to join NoTA developer community by creating a completely new project!

Contents

1. Introduction

NoTA is a new technology for component based platforms. NoTA is service-oriented middleware architecture, which makes service and application development independent of the underlying physical transport layers. Same services can be used on top of TCP/WLAN, Bluetooth, USB or device internal transport channels. Consequently, service developers do not need to know how transport-specific APIs are used.

Different kinds of hardware and software components can be implemented as NoTA services. For instance, NoTA services may provide processing capability, storage capabilities or location information. Also, different input and output components may provide NoTA service interface. One particular application is the keyboard service, which is described in this document. NoTA keyboard service enables different kinds of keyboards (remote or devices' own) to provide input for users' applications.

The keyboard service was developed and demonstrated with device configuration, which is illustrated in Figure 1. Linux laptop device was working as an external keyboard. The laptop hosts NoTA protocol stack and Kbd service node. A Nokia N810 internet tablet with Linux was the target device, whose applications were provided input from the keyboard. The N810 device was running a NoTA application node. Both devices were running NoTA protocol stack. The service was tested using different transports, namely TCP/IP (WLAN), Bluetooth and USB.

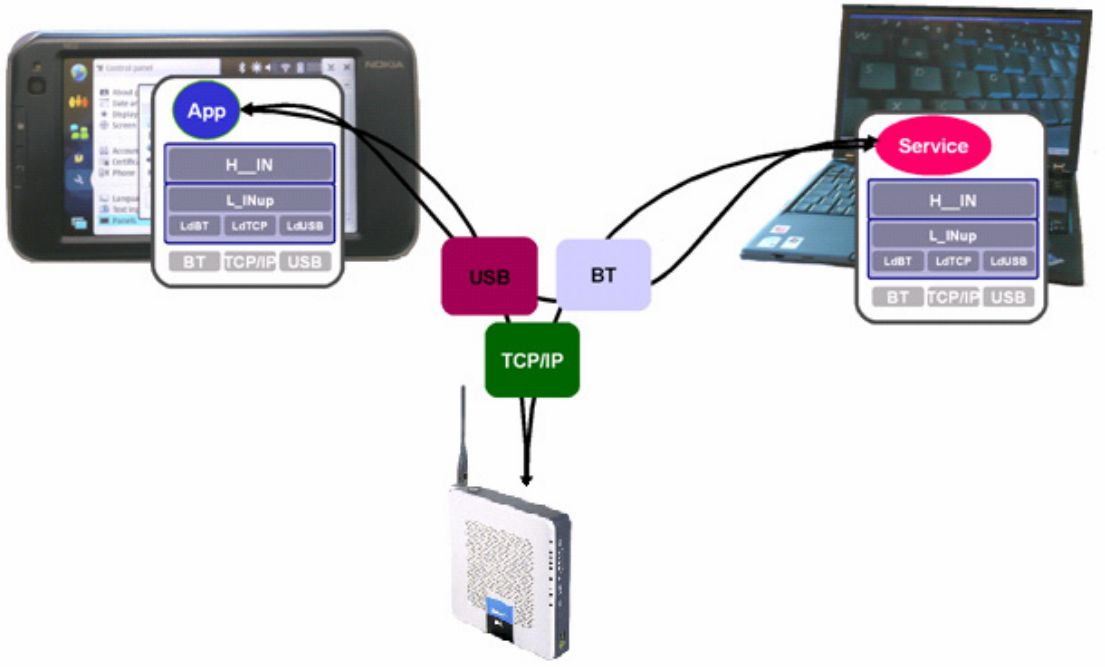

Figure 1. NoTA Keyboard service – components and example deployment to N810 internet tablet and Linux laptop.

2. Distributed keyboards

2.1 Use cases

The demonstrated proof of concept case, illustrated in Figure 1, used laptop's keyboard to control an N810 internet tablet. A typical user migh want to use exteral keyboard, for instance, when writing a longer message or when making some configurations. Using larger keyboard is simply more usable than trying to use the input capabilities available in the small device. Of course, in many cases if there is a better keyboard available there may also be a PC with better messaging program or text editor in that PC already and thus there is no need to use programs in the small device at all.

However, a NoTA keyboard can be used with any kind of NoTA compatible computers. For instance, the user may have only a television and a small standalone keyboard. In this case all applications remain in the television set and the keyboard is running only the keyboard service and a NoTA protocol stack.

NoTA keyboard service is not usable only for remote keyboard cases (i.e. connections over wired or wireless transport). NoTA keyboard service may also provide access to device's own keyboard, which is physically connected to device.

2.2 Requirements for service and underlying layers

Keyboard data consists by nature large amount of small packages, which must be transmitted without delay. There is no large amount of data (i.e. no streaming) and the time when packets come cannot be anticipated. Achieving a feasible user experience sets some requirements for the performance of keyboard service and underlying protocols. Particularly, latency must be small. To achieve this the transport must be ready to send key presses at once. The time when the user presses keys cannot be anticipated. From the NoTA stack point of view this means that the NoTA stack or the used transport protocol cannot go to power saving idle state. Some bluetooth solutions seemed to go into idle mode if there were no transmissions for some period of time. To circumvent this problem, we sent frequent bluetooth pings between test devices to prevent BT from going into the power saving state. However, a better solution is to assure that the underlying transport can be controlled.

To be feasible the keyboard service should be integrated transparently to the system I/O capabilities. Keyboard service should not be usable only with one application but it should be usable with any application (including email, shell, text editor – which ever is active at the time of typing).

Keyboard service sets also some security requirements. Keyboard service can be used to input private and highly secure or critical information such as email messages or passwords. Further, access to the remote keyboard input provides possible attackers an access to shell and, hence, virtually to every application in the victim device. The end user must either utilize security services, which NoTA H_IN will provide,or assure that all transports used by L_IN down are configured to provide a sufficient security level. In the current implementation, there is no security at NoTA-stack level and we assume that the transport security is achieved for example by using WLAN-security mechanisms.

2.3 Related work

There exists different distributed keyboard implementations for different transports. For instance, for N810 linux device there is a Maemo Bluetooth Plugin [1] enabling the usage of Bluetooth keyboards.

Important Linux UI components include the X11 windown system, which provides capabilities to provide keypresses to any active application. Keyboard in Unix/Linux is defined as part of the X system. X is a client-server system to handle display, sound and user input (the whole UI). Also, Linux provides standard utilities for handling keypresses.

3. NoTA overview

Essentially, NoTA consists of APIs and protocols, which hide transport specific APIs and provide transport independent services such as service discovery. NoTA itself is divided into different layers as illustrated in Figure 2. Service and application developers see the high-interconnect (H_IN) layer. Below H_IN there are low interconnect layers (transport independent up (L_INup) and transport specific down (L_INdown)), which enable NoTA to work on top of various transports.

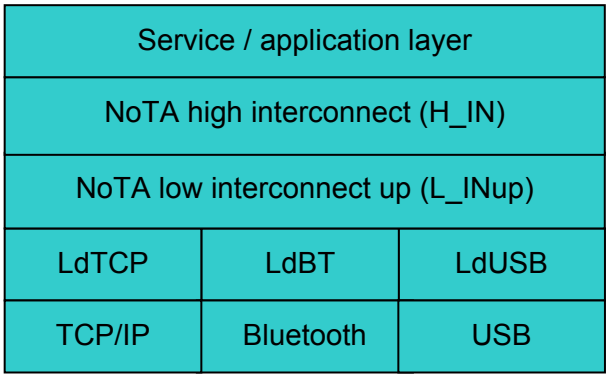

Figure 2. NoTA protocol stack with few transport alternatives.

The NoTA technology is new and being actively developed. Consequently, it utilizes state of-the-art technologies and provides developers a good environment for adding new types of solutions. A downside of new technology is that it still takes some time to mature and reach completely bug-free state.

The technology has been developed to be suitable for small resource limited mobile and embedded devices. However, it is suitable also for any type of devices capable of running NoTA protocol stacks. Currently, NoTA is available for Linux and Windows (via cygwin) devices but other platforms may be supported in the future.

Service developers, typically see the H_IN interface, which is BSD socket [2] like communications interface. Both connectionless (datagrams) and connection-oriented services are available. As in typical socket programming, there are **open**, **close**, **bind**, **connect**, **listen**, **select**, **accept**, **send**, **sendto**, **recv** and **recvfrom** functions available. To ease the application level protocol development, a stub generator program has been released for generating message handling code from XML/WSDL specifications. More information about NoTA can be found from NoTA World web site [3].

It is also possible to use the L_IN interfaces directly instead of the H_IN interfaces. This may provide some performance benefits. L_IN interface provides also transport independent APIs. However, by using L_IN interface directly, services provided by H_IN such as service discovery are not available.

4. NoTA keyboard service implementation

4.1 System architecture

Keyboard service is built on top of NoTA-protocol stack. Keyboard service node (kbdsn) and keyboard application node (kbdan) use H_IN-API functions directly (not through stub generator code). Figure 3 illustrates the communication and the use of API messages in the basic scenario. In the figure, the user first starts both the application and the user node and then presses one key, when the service application's GUI has emerged.

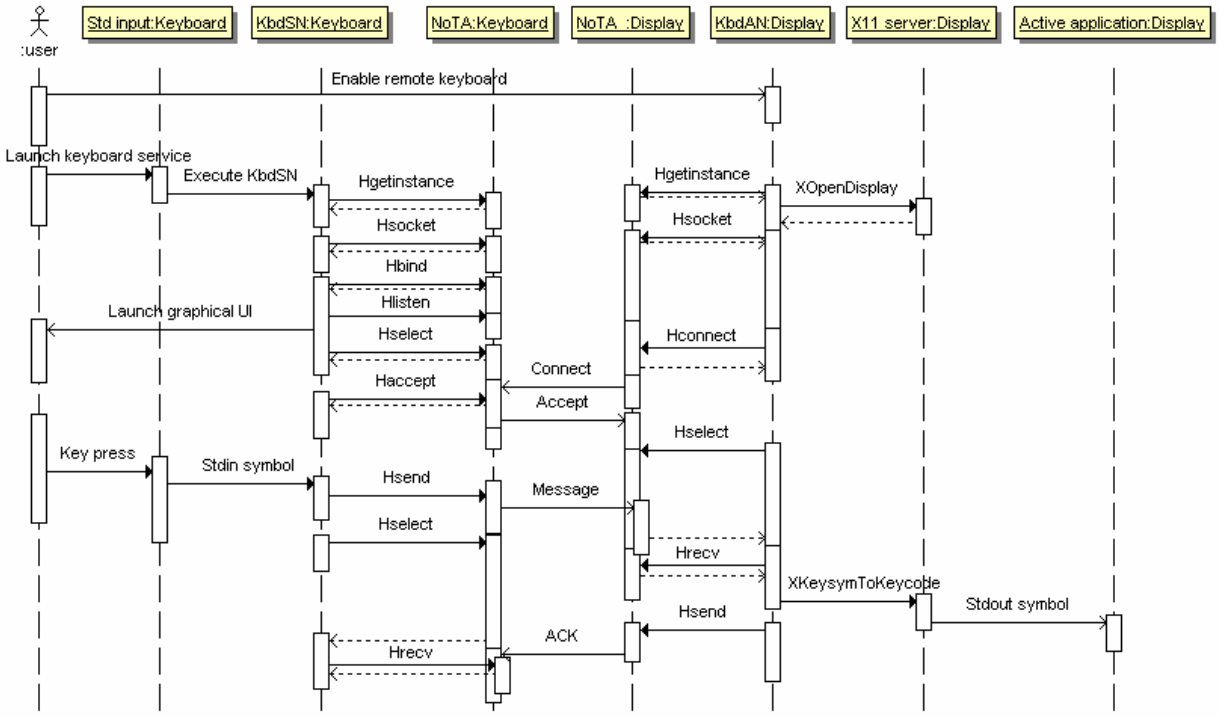

Figure 3. Essential kbd messages.

The main design decisions of Keyboard Service include:

- Two versions of kbdsn and kbdan exists
  - Development time command-line version
  - GUI-version
  - There are different debug-levels
    - NoTA-stack has own debug-options, look at NoTA-stack building (./configure -help)
    - kbdsn prints debug-messages to stdout
    - command line kbdan prints debug-messages to stdout
    - N810 status bar kbdan prints debug-messages to log-file /tmp/kbdan.log

- kbdsn provides server socket where kbdan can connect to
    - Only one client is supported
- kbdan is responsible to maintain connection alive
    - If there is, for any reasons, any connection/send/receive failure socket is closed and kbdan reconnects to service again
- kbdsn reads user's key-presses
    - Command-line version reads characters from stdin
    - GUI-version reads key-presses from GTK-window
- kbdsn converts keys to 32-bit X11-Window System Protocol key symbols
- 4-byte key symbols are sent from kbdsn -> kbdan
- Acknowledgements are sent from kbdan -> kbdsn
    - Message content is the same symbol
- kbdan maintains the connection alive by sending an ALIVE_REQ-message to kbdsn if no data is received within few seconds
    - ALIVE_REQ-message content is 0x00000001
- kbdsn acknowledges the ALIVE_REQ-message by sending an ALIVE_RSP-message to kbdan
    - ALIVE_RSP-message content is 0x00000002
- kbdan simulates key-presses with the Xlib-library.

4.2 Keyboard protocol

Keyboard protocol transmits 4-byte keysyms. On a lower X system level, a keycode value corresponds to a physical keyboard key (A, Ä, SHIFT, ESC etc.). It is for internal use and ambiguous across platforms. To achieve data compatibility between different platforms, the keycodes are converted to standard X system keyboard symbols (keysyms) based on the state of the keyboard. For example, pressing [A] when CAPS-LOCK is on => A (uppercase a). These symbols are platform independent and are used by Xlib functions.

The "X11 Windows System Protocol" standard defines the keysyms (/usr/include/keysymdef.h) for different keyboard variants in different languages. They are actually 29 bit integers but are usually stored into 32 bit (or four byte) integers. (Figure 4.)

```
#define XK_underscore   0x005f  /* U+005F LOW LINE */
#define XK_grave        0x0060  /* U+0060 GRAVE ACCENT */
#define XK_quoteleft    0x0060  /* deprecated */
#define XK_a            0x0061  /* U+0061 LATIN SMALL LETTER A */
#define XK_b            0x0062  /* U+0062 LATIN SMALL LETTER B */
#define XK_c            0x0063  /* U+0063 LATIN SMALL LETTER C */
#define XK_d            0x0064  /* U+0064 LATIN SMALL LETTER D */
#define XK_e            0x0065  /* U+0065 LATIN SMALL LETTER E */
#define XK_f            0x0066  /* U+0066 LATIN SMALL LETTER F */
```

Figure 4. Keysymbols are defined in the file /usr/include/keysymdef.h.

The X system detects both the key press and key release events. However, we currently do not transmit key press or release information so we cannot support key combinations that do not have a symbolic representation (e.g. CTRL-ALT-DEL or CTRL-A do not work but SHIFT-1 should work and produce the **!** character).

4.3 SN

Service node reads characters (from console UI) or receives X keypress events (from GTK+ window UI) in the Linux PC (Figure 5). The node then transforms the characters from internal keycodes into keysymbols and transmits them to AN. In the application node the received keysymbols are transformed back into X keycodes and inserted into the X server.

Notice that the keycode in a Linux PC and an N810 may be different although the keysym is the same!

Figure 5. SN running – graphical user interface on a Linux laptop.

4.4 AN

An application forwards received key symbols to the X window system.

```
#include <X11/Xlib.h>
#include <X11/keysymdef.h>
#include <X11/extensions/XTest.h>
if(display) {
  if(key1 > 0) {
    XTestFakeKeyEvent(display,
                  XKeysymToKeycode(display, key1), True, 0);
  }
  XFlush(display);
}
```

Characters will appear to active window after small delay (AN writes character to a buffer in the X server, which will read and echo them to the active window).

Figure 6. AN user interface – AN is launched from control panel. Icon indicating availability of keyboard service is visible in the top right corner.

GUI version of the application is launched and stopped from the control panel (see Figure 6). The launcher is essentially a shared object library, which is installed to /usr/lib/hildon-status-bar/. Alternatively, the application can be launched from the X terminal (more debug information).

An icon indicating availability of the service is displayed in the status bar (Figure 7).

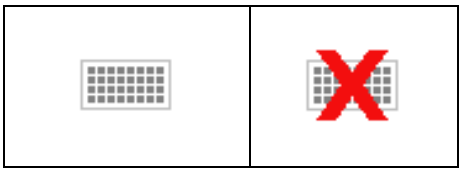

Figure 7. Icons in N810 status bar indicate whether keyboard service is enabled and available or whether it is disabled or unavailable.

To make application able to recover from failures, AN tries to make frequently reconnect when it is not connected.

```
while(alive) {
  if(state != CONNECTED) {
    if(kbdConnect() != 0) {
        if(alive) {
          usleep(1000000);
        }
    }
  }
```

Instructions for building and installing as well as for using NoTA keyboard service can be found from appendices A and B.

5. Conclusions

NoTA is a middleware solution, which eases the development of services in heterogeneous network environments. It provides mechanisms and APIs for service communication and service discovery that are independent of transport technologies and of their imperfections.

NoTA keyboard service provides a development example. It demonstrates that new system-level NoTA services can be developed relatively easy by utilizing existing components.

NoTA development is beginning to gain more momentum. However, there is a need for introducing different services as well as further development in the NoTA stack level. Also, the kbd project welcomes all new developers.

References

1. Maemo Bluetooth Plugin. WWW site. http://770.fs-security.com/maemo-bt-plugin/.

2. Berkeley sockets. Wikipedia entry. http://en.wikipedia.org/wiki/Berkeley_sockets.

3. NoTA World. Open Architecture Initiative. WWW site. http://www.notaworld.org/.

Appendix A: Building and installing

Building libraries and executables

NoTA-stack-libraries (H_IN and L_IN) must be built and installed before keyboard application can be built. Keyboard applications build instructions:

PC:

In the kbd/ directory:

```
$ make clean
$ make
```

Executables kbdsn and kbdan should appear.

N810:

Maemo-Chinook development environment must be installed, more information can be found from http://maemo.org/development/.

```
$ make clean
$ make
```

N810 status bar library libkbdan.so should appear.

Installing KbdAN (GUI version) to Nokia N810 internet tablet

Copy all NoTA-stack-libraries to N810's /usr/lib directory.

Copy kbdan.desktop to N810's /usr/share/applications/hildon-status-bar directory.

Copy libkbdan.so to N810's /usr/lib/hildon-desktop directory.

Appendix B: User guide

Steps below describe how keyboard service is run:

- WLAN-AP: power on
- Laptop: enable WLAN and Bluetooth
- Laptop: start keyboard service at X-terminal
  - $ RM=1 ./kbdsn
  - Keyboard service can be connected through TCP/IP, Bluetooth or USB
- N810: enable WLAN and Bluetooth
- N810: start keyboard application
  - Menu/Settings/Control panel/Panels/Status bar/kbdan
    - Keyboard icon appears to status bar
    - Red cross over icon indicates that keyboard service is not active
      - Activate/deactivate keyboard service by clicking icon
      - Activation/deactivation affects only to the key presses being simulated or not, it has nothing to do with the keyboard service connection
  - Keyboard application connects to the keyboard service
- Laptop: press keys
  - Key presses appear to top window of N810.

ISBN 978-951-38-7168-0 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)