Network Status Support in KDE and How To Use It

Will Stephenson

Akademy 2006, Trinity College Dublin
Overview

• Motivation
• KDE 3 architecture
• KDE 4 architecture
• KDE devices live in dynamic networks
• We should give users and applications a smooth ride in these conditions
• We have architecture to do this
• Listen to find out more
Use cases

• Network Control and Management
  • dialup connections
  • setup WiFi connections
  • Virtual Private Networks/RAS

• Network information to the user
  • feedback on current conditions

• Network information for applications
  • smoothly adapt to network topology
• Network Control & Management, User Information
  • Lots of applications with their own interfaces to the hardware
  • using iwlib or command line tools, (iwconfig, iwlist & co), vendor specific backends
  • KWifiManager, KNemo, Kinternet+smpppd, KPPP, KNetworkManager
• No consistency for KDE
• High maintenance cost in a difficult area
• Network information for applications
  • kded_networkstatus daemon module
  • 2 DCOP interfaces
  • Networking Service interface
  • Client interface
• KDE 3 architecture diagram

- Client applications
- Client DCOP interface
- kded_networkstatus
- Service DCOP interface
- Backends (e.g., smpppd)
• Typical usage
  • Application checks network status
  • Warn user if in offline mode
  • Request online mode (asynchronously)
  • (Daemon requests connection)
  • Queue action
  • On receiving statusChanged signal, proceed
• kio_http
• kopete
• kmail
• GroupWise kdepim KResource
• knetworkmanager
• kded_networkstatus Service Interface
  • registerNetwork( networkName )
  • setNetworkStatus( networkName, status )
  • unregisterNetwork( networkName )
  • requestShutdown( networkName )
• **kded_networkstatus** Client Interface
  • `networks()` - all the registered backends
  • `request( hostname )` a network connection
  • `relinquish()` a network connection
  • `statusChange()` signal
  • `shutdownRequested()` signal
• Problems
  • Mixed connection modes, eg to localhost
  • Initiating connections for specific routes (VPN)
  • No scope for different connection types (WiFi vs GSM modem, call by call)
  • Application uptake could be broader
  • Implementation problems (DCOP per http get)
• Changes
  • Use Solid as common layer for talking to network layers, well structured framework
  • High level backends (NetworkManager) were developed in KDE 3 lifecycle, use them where available
  • platform specific backends (Win/Mac/*nix)
  • More extensible, by using plugins instead of hacking kded_networkstatus
KDE 4 Architecture

- client applications
- kded_networkstatus
- Solid frontend classes
- Solid backend classes
- Backend (e.g., NetworkManager)

DBUS connections:
- kded_networkstatus
- Solid frontend classes
- Solid backend classes
- Backend (e.g., NetworkManager)
• Solid Frontend API
  • class NetworkDevice
    • networks()
    • wireless network appeared signals
  • class Network
    • IP details
    • setActivated( bool )
  • class WirelessNetwork
    • WiFi specifics
    • authentication
• Solid Frontend API
  • class Authentication
    • data for authentication/crypto schemes
  • class NetworkManager
    • main control object
    • access NetworkDevices
    • flight mode
    • disable networking
  • VPN, Dialup (in preparation)
• NetworkManager+dbus backend
• Outreach for other platforms
• Application support library for kded_networkstatus
• Add to KIO::Job?
Conclusion

Thank you!

Questions?

kde-hardware-devel@kde.org
kde-networkmanager@kde.org
(NM, KDE3 specific)