KDE Kontact: An Application Integration Framework
PIM Components Get Together

David Faure
KDE Project
faure@kde.org

Ingo Klöcker
KDE Project
kloecker@kde.org

Tobias König
KDE Project
tokoe@kde.org

Daniel Molkentin
KDE Project
molkentin@kde.org

Zack Rusin
KDE Project
zack@kde.org

Don Sanders
KDE Project
sanders@kde.org

Cornelius Schumacher
KDE Project
schumacher@kde.org

Abstract

Kontact is the new integrated KDE personal information management application. Well, it seems new, but in fact only the shiny surface is new. Under the hood well-known time-honored KDE applications like KMail, KOrganizer, KAddressBook, KNotes and KNode do their work. This paper describes how KDE component technologies like KParts, DCOP or XMLGUI enable embedding of applications to create something which is more than the sum of the parts.

1 Introduction

Kontact [1] is a new KDE [2] application for managing personal information like mail, appointments, todo lists and contacts. It is based on existing KDE applications which are embedded into a container framework by using KDE component technology and extending it where required. Figure 7 shows a screenshot of the organizer component in Kontact.

The functionality of Kontact emerges from integration of the components on the application level. It doesn't compromise the ability of the components to run as individual stand-alone applications. The user retains the choice whether to use the components as one application aggregating all the functionality or to use some or all parts as separate applications.

KDE is one of the major desktop environments for Linux and Unix systems. it is based on the Qt toolkit [29] and mostly written in C++. It consists of a framework providing the desktop infrastructure and a wide range of applications, from web browser and mail client, through music players, games and educational software to text editor and integrated development environment. Kontact is part of the kdepim module of KDE which contains tools for personal information management.

After a short discussion about components and monolithism (section 2) this paper will give an introduction into the Kontact component model (section 3) and a detailed overview of the integration technologies used in Kontact (section 4). It will discuss the importance of open standards (section 5), give an insight into Kontact as project (section 6) and will conclude with information about the availability of Kontact (section 7), final remarks (section 8) and some information about the authors of Kontact (section 9).

2 Components vs. Monolithism

Why bother with integrating originally disparate applications? Why not build a monolithic integrated application from ground up with all its parts fitting together from the beginning?

The technical advantages are that a loose coupling forces modularization and cleaner interface definitions. This leads to a better factored architecture which is inherently easier to maintain and leverages reuse of existing code.

On an economical side this kind of reuse saves the investments, in particular of time and effort of the open source community, in the already existing applications. All the components integrated into Kontact have a long history as independent applications and provide a stable and feature-rich base for the integrated framework.

There are also social reasons. Each of the different Kontact component applications have their own healthy developer communities. By providing a technical integration framework on the top of those applications the developers also get integrated. They still maintain their application development environment but gain a new sense of community on a higher level. The Kontact experience shows that this works surprisingly well and gives room for new synergies. The monolithic approaches we have seen in the past in this area have suffered from the not-invented-here syndrome and reinvention-of-the-wheel scenarios. There are several projects which tried to write a monolithic application comparable to Kontact and failed because of these reasons.

On a philosophical level the integration of GUI desktop applications as done by Kontact can also be seen as interesting parallel to the classic philosophy of UNIX commands, single-purpose commands which can be assembled to complex and powerful aggregations to fulfill almost any task imaginable. By combining single-purpose applications in an integrated suite for personal information management this philosophy is raised to the application level.

3 The Kontact Component Model

The Kontact framework acts as a container for plugging in other applications as components. It provides the standard environment like main window, menu, tool and status bars as well as a navigation bar to control the embedded components. Applications to be embedded into this environment have to be accompanied by a Kontact plugin. The plugin acts as mediator between the framework and the application. It exposes the functionality of the application which is being integrated by implementing the Kontact Plugin API and interacts with the Kontact Framework by using the Kontact Core API. All this is done in-process. Communication between the components is done via DCOP through clearly defined standard interfaces. Figure 1 shows the component model.

The plugin API specifies functions which have to be implemented by concrete plugins for providing access to the corresponding KPart object, the summary view and options how the component appears and is handled in the framework application. This includes title labels, icons and hints about the position in the navigation bar. In addition to that it provides functions to control the component behavior, starting and selecting of a component, requesting if the component runs embedded or stand-alone and access to the interfaces of the underlying component technologies. Finally the plugin API provides functions for handling of drag and drop functionality and user interface actions.

The framework core API which gives the plugins access to the framework mainly deals with loading and selecting of plugins belonging to other components. Inter-component communication is done by using the DCOP interfaces of the individual components.

A Kontact plugin can provide an application or document main view, a summary view and some specific functionality and data like menu and toolbar actions, configuration, ”about” data and more. If an application already provides a KPart as part of its interface adding a Kontact plugin in order to integrate it into Kontact is only a few lines of code.

4 Application Integration Technologies

The KDE framework provides a rich variety of integration technologies [5]. They have been used on a component level for applications like the KDE web browser Konqueror [14] and on a slightly higher level in the KDE office suite KOffice [15]. Kontact constitutes the integration on the highest level - the level of complete applications.

This section will present the KDE component technology KParts (section 4.1), the desktop communication protocol DCOP (section 4.2), the user interface description framework XMLGUI (section 4.3), the integrated configuration framework (section 4.4), and the KDE resources framework KResources (section 4.5).

4.1 KParts

KParts is the KDE component technology introduced with Konqueror and KOffice. A KPart is a dynamically loadable module which provides an embeddable document or control view including associated menu and toolbar actions. A broker returns KPart objects for certain data or service types to the requesting application. KParts are for example used for embedding an image viewer into the web browser or for embedding a spread sheet object into the word processor.

KPart instrumentation of an application is a low-effort task, because all the technologies used are usually already utilized in the existing code. There are no new programming languages, external processes or protocols involved. It basically boils down to the implementation of a specific C++ interface. Kontact components all have a KPartification history of heroic one-weekend or even one-afternoon hacks.

4.1.1 What’s a KPart?

KParts consist of a view object embeddable in a user interface frame, associated actions like menus items and toolbar buttons and optional extension objects for handling the status bar or special functionality like the forward, backward and history functions of a browser. They are accompanied by metadata like author and copyright information and service types the KPart provides. The metadata is provided as files adhering to the Desktop Entry Standard [35]. This standard, hosted by freedesktop.org, is used in many other places in KDE and other desktop environments, e.g. GNOME [3], to store metadata.

The view object provided by KParts is a standard user interface component which is usually embedded as main component into the main window of the embedding application. It represents the view or main work area of the functionality provided by the KPart.

In addition to the view the KPart provides user interface actions which correspond to menu items or toolbar buttons. The menus and toolbars of KParts are combined with the menus and toolbars of the main window or other components. The actions are described and created based on an XML descriptions of the available actions. This mechanism is known as XMLGUI. It is de-

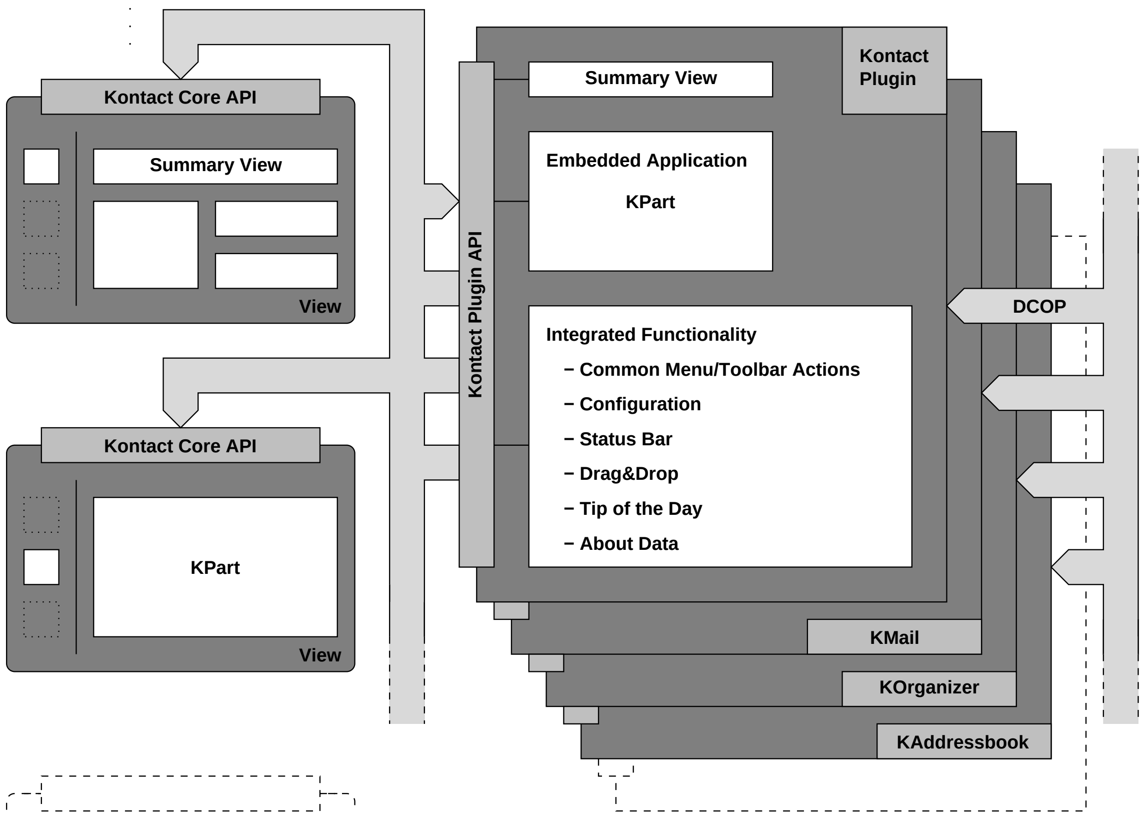


Figure 1: Kontact Component Model

scribed in more detail in section 4.3.

Special functionality of the embedding application for use by the KPart can be provided by extension objects which are created by the KPart and made externally visible. One example is the status bar extension which gives KParts access to the status bar of the main window of the embedding application. The KPart sends the messages to be shown in the status bar to the extension and the embedding application takes care of showing them and synchronizing the status bar information when different KParts are activated.

4.1.2 KParts Framework

KParts are loaded as dynamically loadable modules at run-time. A trader mechanism based on the service types provided by the KParts is used to select the KPart which fits best to the data or application to be handled and the preferences of the user. For example in Kontact this mechanism is used to find the applications which provide the Kontact integration, and in Konqueror it is used to find appropriate viewer and editor parts for handling documents of specific MIME types.

The actual KPart API basically consists of the functions to select, load and find the desired KPart instance. Specific functionality of the KPart instances is realized by subclassing. The document handling KParts used in applications like Konqueror, for example, provide an interface to load and save documents in a network-transparent way in the classes `ReadOnlyPart` and `ReadWritePart` [7].

Communication between KParts is done via the Desktop Communication Protocol (see section 4.2). This extends the API of the specific KParts by functions not only available to the embedding framework but also available to other components within or out of the process of the embedding application.

The KPart interfaces don't include remote function calls, unlike many other component technology frameworks like DCOM [8] from Microsoft, UNO [10] from OpenOffice.org or Bonobo [11] from GNOME. Historically the predecessor of the KParts framework which was based on CORBA [9] from The Open Group did so, but due to its complexity it wasn't accepted by application developers and it was eventually replaced by the current combination of KParts and DCOP which provides a simpler framework for use within a desktop environment. For a more detailed description of the KParts component architecture including a comparison with other models, especially CORBA based models see [6].

4.1.3 Use of KParts in Kontact

The classical KParts model originates from document-oriented views in KOffice and Konqueror. The difference in Kontact is that the KPart instances are not associated to specific documents, but represent views to more global objects, like the user's mails, calendar or contact data. Kontact doesn't subclass the KPart's API, but uses DCOP calls to access the specific functionality of the embedded applications.

Kontact loads a KPart object for each application it embeds. The KPart provides the main view which is usually used in the main window of the stand-alone version of the application. Loading is done on demand, so that only those objects consume memory and affect startup time which are actually used. Unlike in purely document-oriented user interfaces multiple KPart instances are loaded in parallel and communicate with each other, reflecting the situation of multiple applications running in parallel and working together. Concerning the user interface there still is a primary KPart instance which is active. This means that its view is shown in the main window and its part-specific actions are merged into the applications menus and toolbars.

4.2 Desktop Communication Protocol (DCOP)

The Desktop Communication Protocol (DCOP) is the well-known KDE inter-process communication mechanism. It is based on a central server relaying function calls between applications. DCOP makes use of the Qt object serialization implementation and uses the Inter Client Exchange (ICE) protocol [12] (part of X11R6) as transport layer.

The key feature of DCOP that makes it a viable choice for Kontact's inter-component communication is that it is also able to make in-process calls and thus minimizes the overhead for communication inside the one-process Kontact component assembly.

As this is done completely transparently to the sender and receiver of DCOP calls, it makes it possible to either run Kontact components inside of the Kontact framework or as stand-alone applications without any difference to the user other than the additional GUI integration inside the Kontact framework and the features this inherently adds, e.g the integrated configuration, the summary view or the extended drag and drop support. This gives users the choice to run the application in a way that best fits to their personal working style.

4.2.1 DCOP Implementation

DCOP communication between processes is mediated by a special server process. This server is started with each desktop session and handles all DCOP communication between applications belonging to this session. When an application wants to talk to another application it sends the request to the DCOP server which in turn forwards the request to its destination. Responses are sent back to the server which returns them to the application from which the request originated. The fact that communication goes through a server is completely transparent to the applications using DCOP.

When an application provides a DCOP interface for use by other processes it registers itself with the application name with the DCOP server. In addition it registers each DCOP interface it provides with a specific name with the server. DCOP provides inspection functions so that it is possible to browse DCOP interfaces in order to find out which interfaces are registered and which functions they provide.

Direct function calls are not the only way to use DCOP. it is also possible to use a signaling mechanism where an application registers for a specific signal and the server then notifies the application when the signal is emitted by the application which was requested. By using wild card matching flexible control over which signals an application listens to is possible.

Actual communication in DCOP is done using the ICE library, part of the X server, as transport layer. The communication is done by using Unix sockets. DCOP intentionally doesn't provide network transparent communication because this isn't required in normal desktop scenarios and would add an additional level of complexity as well as imposing a new class of security problems which don't need to be handled if network access is disabled.

Serialization and deserialization of data is done by using the C++ stream operators associated with all the basic data types in the Qt toolkit. They use a compact binary format. This is hidden from the DCOP user. Only if new or custom datatypes are to be sent over DCOP do new stream operators have to be implemented. This is mostly very easy because it can be based on existing operators for the data types the new type is composed of.

DCOP interface compiler For convenient instrumentation of an application with DCOP interfaces there is a special tool, the dcopidl compiler. It takes a file describing the interface and generates the C++ code required to implement the interface. The interface description file is basically a C++ class header with some additional keywords to identify the functions for instrumentation by DCOP. So usually it is enough to add these keywords to the already existing header declaring the functions to be available via DCOP and run the dcopidl compiler on this file. For normal compilation the special keywords are defined to be empty so that they are ignored by the compiler. Figure 2 shows an example of a header used

```
class KCalendarIface : public DCOPObject
{
    K_DCOP
  public:
    KCalendarIface()
      : DCOPObject("CalendarIface") {}

  k_dcop:
    virtual void showTodoView() = 0;
    virtual void showEventView() = 0;

    virtual void goDate( QDate date ) = 0;
    virtual void goDate( QString date ) = 0;
};
```

Figure 2: Example code providing a DCOP interface

to generate a DCOP interface.

The dcopidl compiler also provides the capability to generate stub classes which can be used to make DCOP function calls. This way the DCOP call appears as normal type-safe function call to the calling application and the DCOP communication and the fact that the call actually is a remote function call becomes completely transparent to the applications using DCOP.

The KDE build system automates running the dcopidl compiler so that DCOP interfaces can be added or used by adding only a few lines to the Makefile templates or header files.

The abstraction introduced by the dcopidl compiler could be used to replace DCOP by another transport mechanism without any changes to applications simply by modifying the dcopidl compiler to generate stubs and interface implementations for another transport mechanism. This would be one way to add support for other inter-process communication mechanisms like D-BUS [13] or platform-specific technologies.

Application scripting DCOP is not limited to communication between applications. It can also be used by end users to control an application remotely. There are several different methods available to do this conveniently. For example, one can use the `dcop` command line tool for sending DCOP requests, the corresponding GUI tool `kdcop`, or use the language bindings to script the application in languages like Perl or Python.

4.2.2 Use of DCOP in Kontact

For Kontact DCOP is the main communication mechanism between components. It is used for inter-application communication which, depending on the configuration of Kontact plugins, means interacting with applications running as external processes with their own user interface or running embedded into Kontact in-process. DCOP handles in-process calls in a special way without using the dcop server process in order to optimize the efficiency of DCOP calls.

To maximize flexibility most of the interfaces used for communication between components are not tightly coupled to the implementation of the component. There are abstract interfaces for services like ”email client” or ”organizer” which are implemented by specific applications like KMail or KOrganizer. In principle these services could also be implemented by other applications, so that the user could choose the specific application for being used in Kontact. This might even be used to integrate non-KDE applications by adding a small DCOP wrapper around the specific interfaces of the application to be integrated.

As components are loaded on demand and stand-alone applications might not yet have been started when another component requests communication there is a special service for starting DCOP services on demand. The component is then started according to the users configuration inside Kontact or as stand-alone process when a request to the service interface implemented by the component is initiated.

One problem that arises with components being optionally able to run stand-alone or embedded into the Kontact framework application is that the name of the application registering the DCOP interfaces is different for these two cases. Thus components inside of Kontact register twice, once with the name of the container application and a second time with the name of the stand-alone application, so that the interfaces are always available under the same name.

Unique Applications DCOP is also used for realizing so-called ”unique applications”. These are applications which can only be started once per session. This is for example used to make sure that processing of mail folders, calendar or address book data always takes place in a single process in order to avoid problems with concurrent access to the same data by different instances of an application. A unique application has a simple standard DCOP interface. When a unique application is started it first looks if a process already has registered this standard DCOP interface under the name of the application. If the process already exists it is called to handle the request to start another instance. This usually means that an existing main window is activated or that command line options are processed. If the process doesn't exist yet application startup proceeds and registers the DCOP interface, so that it is visible for subsequent starting of application instances.

The standard unique application handling isn't completely sufficient for the case of Kontact, because an application can be run in two ways, as stand-alone application where the standard unique application handling applies and as embedded component where the process providing the unique DCOP interface is different from the stand-alone case. A DCOP watcher class handles this situation by forwarding the unique application DCOP requests to the correct destination. This makes sure that for example when calling KOrganizer from the command line and a KOrganizer component in Kontact is already running the component in Kontact is activated instead of running a second instance stand-alone.

Hidden DCOP functions Kontact also makes use of a special feature of the dcopidl compiler which allows developers to hide DCOP functions from the standard interface inspection provided by the dcop and kdcop tools. This makes it possible to hide parts of the DCOP interfaces, so that they can be used for inter-component communication, but are not available to tools like dcop and kdcop which rely on the interface inspection capabilities of DCOP.

4.3 XMLGUI

The integration of separate applications into a common main window requires merging of menus and toolbars. KDE provides an abstract way to define menu and toolbar actions using XML descriptions of where the actions are placed. These descriptions are loaded and processed at run-time. This allows to change menus and toolbars to be changed without changing code. It also provides a way to manipulate menus and toolbars programatically from outside the code implementing the actions.

This flexibility is a key requirement for embedding actions from different applications in a single framework. It allows to menus to change dynamically and toolbars to reflect the functionality provided by the currently selected component. In addition to that it allows components to inject additional actions which can optionally be available independently of the selected component and it makes it possible for the framework to remove actions which would be redundant inside the integrated application. Examples are configuration options, ”new” actions and ”about” dialogs, which are all provided by global actions of the container.

The menu and toolbar parts of KParts (see section 4.1) are based on XMLGUI. Figure 3 shows an example of an XML user interface action description.

One advantage of the XMLGUI mechanism is that it makes it possible to provide a general configuration mechanism for the actions to the user. There is a standard configuration dialog which enables the user to persistently change which actions are shown as toolbar buttons.

Another advantage is that a it is possible to change menus and toolbars of an application by just changing or providing more specific action descriptions. This can be used by system administrators to lock down applications and make parts of the functionality unavailable to users.

```
<!DOCTYPE kpartgui >
<kpartgui version="15" name="kontact" >
  <MenuBar>
    <Menu name="file">
      <text>&amp;File</text>
      <Action name="action_new"/>
      <Separator/>
      <Action name="file_quit"/>
    </Menu>
    <Menu name="settings">
      <text>&amp;Settings</text>
      <Merge append="save_merge"/>
      <Action name="settings_configure" />
    </Menu>
  </MenuBar>
  <ToolBar name="mainToolBar">
    <text>Main Toolbar</text>
    <Action name="action_new"/>
    <Merge/>
    <Action name="help_whats_this"/>
  </ToolBar>
</kpartgui>
```

Figure 3: Example XML description of user interface actions

This is used by the KIOSK mode described in section 4.4.3.

4.4 Integrated Configuration

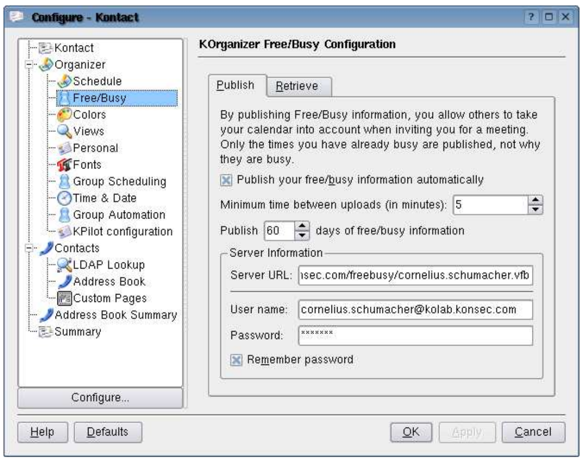
Configuration of Kontact is done by extending the modular mechanisms used in the desktop-wide KDE control center to the application level. Configuration dialogs are composed of modules which are dynamically loaded at run-time. A generic configuration dialog provides access to all modules relevant for the applications embedded into Kontact.

The selection of the components which are present in Kontact is done by an extension of the metadata associated with all programs. The generic KDE plugin selection infrastructure makes use of this data by providing a backend for accessing activation state of plugins and a GUI for the user selecting which plugins should be active.

General access to the actual configuration options is performed through the KDE configuration backend (KConfig XT, see section 4.4.2). It provides an abstract description of configuration options which is reused in the GUI, a generated API to the configuration data which is used to access and share configuration data in a well-defined way, and other things, like the desktop lockdown features of the KIOSK mode.

4.4.1 Common Configuration Dialog

The configuration dialog is one example where application integration in Kontact is more than just merging views in a common main window. The configuration dialog provides access to all options of the components se-



KOrganizer Free/Busy Configuration

By publishing Free/Busy information, you allow others to take your calendar into account when inviting you for a meeting. Only the times you have already busy are published, not why they are busy.

[✓] Publish your free/busy information automatically

Minimum time between uploads (in minutes): 5

Publish 60 days of free/busy information

Server Information

Server URL: ısec.com/freebusy/cornelius.schumacher.vfb

User name: cornelius.schumacher@kolab.konsec.com

Password: *******

[✓] Remember password

Figure 4: The Kontact configuration dialog

lected to be embedded into Kontact. It creates a common hierarchical view of all available configuration pages. See figure 4 for a screenshot of the integrated configuration dialog.

Technically the different pages of the configuration dialog are implemented as dynamically loaded modules using the same programming interfaces as the desktop-wide KDE control center. Each module has an associated file which contains meta and control information like the name of the module including translations, the name of the library to load or the type of control module, i.e. in which dialog and where in the hierarchy it is to be shown. This file uses the Desktop Entry Standard [35]. This mechanism allows the construction of the navigation part of the dialog without actually loading the modules. The fact that the modules are implemented as independent modules allows them to be loaded on demand as they are needed by the user interface, even without requiring to have the KPart of the associated application to be loaded in memory.

4.4.2 KConfig XT

For storage of configuration data the standard KDE configuration backend is used. It stores the data as grouped key-value pairs in simple files using an INI-style syntax. The configuration backend supports cascading configuration files, thanks to which users settings are read and merged from several files, starting from global system-wide files down to user-specific files in the users home directory. In general information from more specific files takes precedence over information from more global files, but this can be modified in the KIOSK mode.

KConfig XT (XT stands for ”extended technology”) is an additional level of abstraction on top of the standard configuration backend. it is based on an abstract description of the configuration using XML. See figure 5 for an example file. The description contains information on the names and types of configuration entries, the text used for labeling and explaining the options, default values and a logical grouping of the entries. It also contains some control information like where the configuration is stored.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE kcfg SYSTEM "http://www.kde.org/
  standards/kcfg/1.0/kcfg.dtd">
<kcfg>
  <kcfgfile name="kontactrc"/>

  <group name="View">
    <entry type="String" name="ActivePlugin">
      <default>kontact_summaryplugin</default>
    </entry>
  </group>

</kcfg>
```

Figure 5: Example XML file used by KConfig XT for describing configuration options

Configuration Code Generator To provide applications with convenient and type-safe access to the configuration data KConfig XT includes a tool for generating the C++ code, required to access the data, from the XML descriptions files: the ”kconfig compiler”. The generated code provides direct access to the configuration entries by creating individual functions for each entry. These can be used within the application code when access to specific entries is needed. There are various options to control the generation of the code, so that it can be adapted to the specific needs of applications. See figure 6 for an example of generated code.

The generated classes also provide a generic way to access the configuration entries. This is used for widely used actions like reading and writing of the configuration data. The generated class is also used for connecting configuration dialogs to the backend in a generic way. There is a special class `KConfigDialog` which associates the widgets of a configuration with configuration entries based on the names of the entries and the widgets. This makes it very easy to create configuration dialogs because all handling of the actual data is done automatically in the backend and the developer only has to create an XML description of the configuration options and a corresponding dialog. If this is done with GUI design tools like Qt Designer [30] creating configuration dialogs can be done without having to write more than a few lines of code.

Tools There are some additional tools under development which make use of the generic nature of the KConfig XT framework. One is an application-independent editor for configuration options which works on the configuration files via KConfig XT. This allows to manipulate options which have no representation in the GUI, to edit remote configuration files or to edit the configuration of multiple applications in one go. The other tool under development is a graphical editor for the KConfig XT XML descriptions. This makes it possible to create and edit configuration without having to know about the XML format. Together with Qt designer this makes a powerful suite of tools for easy implementation of the configuration parts of applications.

Benefi ts The advantages of the KConfig XT approach are that there is only a single location where configuration keys and default values are specified, that there is type-safe access to the data for applications, that details of reading and writing the configuration, e.g. applying default values, are automatically handled and that there is a way to access the configuration in an abstract way, which is in particular useful for generic tools not tied to a specific set of configuration data, for example a generic configuration dialog.

Kontact which, by its nature of application integration framework, combines a lot of configuration options greatly benefits from the simplifications which are made possible by the use of KConfig XT.

4.4.3 KIOSK Mode

To fine-tune the configurability of applications and to give administrators a tool to control how users configure their applications KDE provides the so-called KIOSK mode. This is an extension of the standard configuration backend which allows to control how configuration options can be changed. This offers all the tools required for features like desktop-lockdown.

The key concept of the KIOSK mode are immutable options. Any entry of a configuration file can be made immutable by adding a special flag to its key. This means that the configuration backend won't write any changes of this entry anymore. As configuration files are read in a cascaded way an administrator can add the flag making an entry immutable in a global configuration file which the user doesn't have permissions to write. This way the option can't be changed by the user anymore.

4.4.4 Configuration Wizards

With the integration of different applications in a common framework, as happens in Kontact, the need for common configuration increases. There are several configuration options which are the same for different applications, like name and email address of the user, depend on each other or can be deduced from other options, like server names, addresses of certain files or services on the same server.

One example is the configuration for access to a groupware server where data like addresses of incom-

```
namespace Kontact {

class Prefs : public KConfigSkeleton
{
  public:
    static Prefs *self();
    ~Prefs();

    /**
      Set ActivePlugin
    */
    static
    void setActivePlugin( const QString & v )
    {
      if (!self()->
              isImmutable( "ActivePlugin" ))
        self()->mActivePlugin = v;
    }

    /**
      Get ActivePlugin
    */
    static
    QString activePlugin()
    {
      return self()->mActivePlugin;
    }

    /**
      Get Item object corresponding to
      ActivePlugin()
    */
    ItemString *activePluginItem()
    {
      return mActivePluginItem;
    }
};

}
```

Figure 6: Code generated from the example XML of figure 5

ing and outgoing mail server or for accessing contact or calendar data can be created from knowing which kind of server is to be used together with address and login information for a specific server.

Configuring this information in all the different applications is cumbersome, although the fine-grained configuration is needed for tuning the applications to special needs and to be able to handle different usage scenarios. The classical solution to this problem is to provide configuration wizards which collect the required information for configuration at a centralized location.

Wizard Rules In Kontact this problem is addressed by a special kind of application-spanning configuration wizard based on an extension of KConfig XT for propagation of configuration options. The information which is needed to deduce the detailed configuration for the individual applications is described by a standard KConfig XT XML file and the corresponding GUI is created on top of this. For propagation of the information to the configuration of the individual applications a set of rules is added to the KConfig XT XML file. These rules can specify simple copying of data to other configuration files or more complex conditional propagation based on other data, e.g. based on the information if a special feature is enabled or not. it is also possible to define custom rules which are accompanied by C++ code. This gives the flexibility to also handle very complex cases.

Wizard User Interface The wizard dialog reads and interprets the data and rules from the description files, integrates the GUI for setting the options and applies the data put in by the user according to the defined rules to the configuration of the involved applications. This mechanism makes it very easy to set up configuration wizards as the application developer can use the generic mechanisms and only needs to write code for handling of special cases. It has the additional advantage that it creates a formal specification of how the configuration is affected by the wizard which can for example be exploited in the GUI to indicate how the different configuration options depend on each other without requiring any special handling by the application developer.

Kontact provides wizards for configuring access for groupware servers like Kolab [31] or eGroupware [32]. It would also be possible to use the wizard infrastructure for setting up user profiles or to make it possible to apply policies to the configuration. It might also be useful for storing and reapplying certain configurations of individual users, e.g. when moving between different systems.

4.5 KResources Framework

The way an application like Kontact accesses its data shows some common patterns for different data types. For example calendar data and address book data are accessed in a similar way. It can be stored local or remote, it can be stored in text files of different formats or in databases-like systems, there can be different sources of the same kind of data, which need different configuration, data can be read-write or read-only, etc.

To address these problems in a common and consistent way and to avoid duplication of code and effort Kontact makes use of the so-called KResources framework which provides an abstract interface for management of data resources. This framework specifies an API for data resources and the user interface to configure these resources. The resource API includes functions for saving and restoring configuration, for opening and closing of resources, for loading and saving data, for naming resources and for handling write permissions. The framework also provides a management class for handling creation, modification, configuration and persistance of KResources objects and a common configuration module which operates on the abstract API of the resources and so provides a user interface for management of all different resource objects at a central place. Usually there is only one set of resources per data type which represents for example a central calendar or address book for a user shared by different applications or Kontact components.

Resource Families Deriving from the abstract interface there is a set of classes defining resources for a certain type of data. They are called families in the KResources framework. Currently there are families for example for calendars and address books.

For the different families there are various implementations of concrete resources, e.g. file based calendars using the iCalendar format, address books accessed via LDAP or resources accessing calendar and address book data on a Kolab server. There are also a bit more exotic resources like one that provides entries of a Bugzilla [33] based bug-tracking system as todo list in KOrganizer.

All KResources are implemented as plugins and loaded at run-time, so that memory consumption of Kontact isn't affected by unused resources and dependencies on special external libraries are isolated in the plugins without adding to the dependencies of the Kontact framework application or other components.

Change Notifi cation The KResources framework also includes a mechanism to notify different instances accessing the same set of resources about changes in the resource configuration. All resource management objects running in the same desktop session communicate with each other for this purpose, either in-process or between different processes by using DCOP. So if a new calendar file is added to the user's calendar in the central resource management configuration module it automatically appears in the calendar view of Kontact, regardless of whether the calendar runs embedded in Kontact or as stand-alone application.

5 Open Standards

One important aspect of interoperability between applications are open standards for file formats, network protocols and other ways of interaction or data exchange. This is important for interoperation of components inside of Kontact as well as in a wider context of interoperation with all kind of applications and servers from different provenience. The fact that the standards used are open is of particular importance for Free Software projects like Kontact to ensure that specifications and other information is available to all developers and the resulting code is free for distribution under Free Software licenses.

Kontact makes use of a wide variety of open standards. It implements many of the mail-related RFCs, including POP [18], IMAP [19] and SMTP [20]. The address book component uses vCard [21][22] as storage and exchanges format, and provides support for LDAP [23] as an access protocol. Calendaring is based on iCalendar [24] and the associated group scheduling standards iMIP [26] and iTIP [25].

6 Kontact as Project

In addition to the technical aspects of Kontact it is also interesting how Kontact evolved as a project and how development works in terms of social, organizational and political aspects.

6.1 History

The history of Kontact is much longer than the history of the current project known under the name Kontact. It evolved along with the technologies it uses and together with the community around the components it integrates.

KMail and KOrganizer were part of KDE almost from the beginning, as separate applications. When development for KDE 2 began the kdepim module was introduced. This started as playground for a reimplementation of the KDE address book which then became the KAddressBook of today and for an experimental new mail client called ”Empath”. KOrganizer moved into the kdepim module shortly before KDE 2.0 which was released in October 2000.

The first implementation of a KParts-based framework that aimed at integrating various existing components of personal information management software appeared in the KDE CVS on March 22th 2000. It integrated Empath and KOrganizer but never got to a state where it really did something useful.

In April 2002 the initial version of the Kontact framework was imported under the name Kaplan into the KDE CVS. It was the result of a weekend-hack inspired by the very modular plugin structure of what now is KDevelop 3 [36], which was at that time a brand-new rewrite of KDevelop 2. It integrated KOrganizer and KAddressBook. Shortly after that KMail was also integrated, and the resulting combination was released as Kontact for the first time as a stable preview release.

At the beginning of 2003 after a meeting of many of the core kdepim developers in Osnabrueck, Germany, KMail was moved into the kdepim CVS module, now combining all major components of Kontact in one module. KDE 3.2, in the beginning of 2004, was the first KDE release which included Kontact. This was version 0.8 and already provided a full set of features, mainly due to the long history of its components.

Since then kdepim is working towards the next release which will be Kontact 1.0. One interesting aspect is that the development communities of the different applications Kontact integrates also got integrated and now form a robust and powerful team which, just as Kontact itself, is more than the sum of its parts.

6.2 The Inner Workings of the Kontact Team

Kontact is a classical Free Software project. It has contributors all over the world with different backgrounds and interests but with the common goal to work on Kontact and make it a good application.

Communication and coordination between developers is one crucial point of the project. Many different tools are used for this purpose, e.g. CVS, Bugzilla, IRC, mailing list, web pages, Wikis and more. The central place for communication is the kde-pim mailing list which has around 400 subscribers. More informal discussions as well as communication among developers working together for example during bug fixing sessions happen on IRC. Because contributors are distributed over many different time zones there is activity in the Kontact IRC channel almost 24 hours a day.

Personal meetings have become increasingly important and shown to be essential for efficient discussion of technical questions or common work on code as well as for fostering the relationships between the developers and increasing motivation and community feeling. Usually developers meet on trade shows or on the regular yearly KDE-wide meetings, but in the past year and a half dedicated meetings of the group of kdepim developers have also been established.

6.3 Commercial Improvement System

Most contributors of Kontact do their work in their free time without getting paid for it. This inevitably leads to conflicts with other engagements, be it the job, studying, or social activities. Getting paid for working on Kontact might alleviate these conflicts. In addition to that there are often people who would like to contribute to Free Software projects by spending some money. This is often difficult, because there is no infrastructure for receiving and distributing money.

For Kontact there is an attempt to provide a solution for this problem by offering a commercial improvement system [34]. This could be seen as continuation of the voting system of Bugzilla [33]. The concept is that people willing to spend some money for implementation of certain features in Kontact pledge an amount of money on the specific feature they would like to see implemented. Developers prioritize their work oriented at the accumulated amount of money per feature and work on the features with the highest amount. Once the feature is completed and the people pledging the money are satisfied with the result, they actually transfer the money to the developers having implemented the feature.

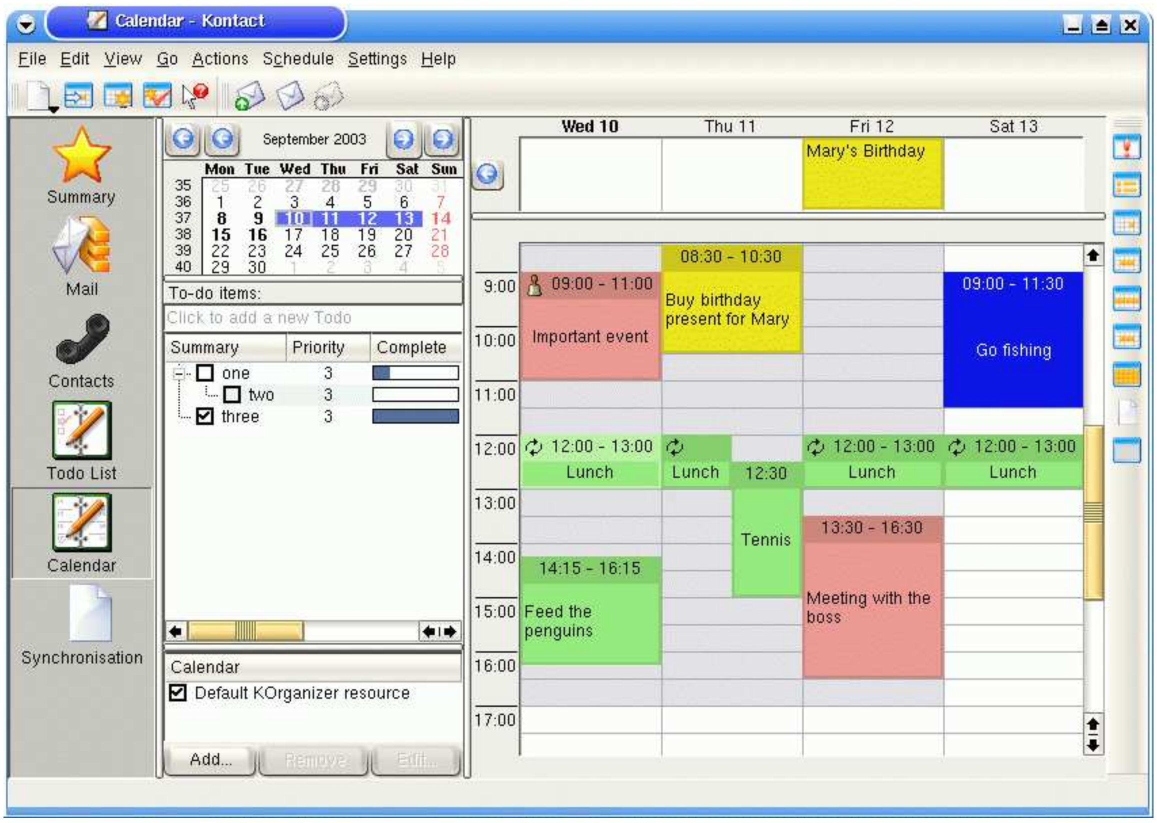

Calendar - Kontact
File Edit View Go Actions Schedule Settings Help
Summary
Mail
Contacts
Todo List
Calendar
Synchronisation
September 2003
Mon Tue Wed Thu Fri Sat Sun
To-do items:
Click to add a new Todo
Summary Priority Complete
one 3
two 3
three 3
Calendar
Default KOrganizer resource
Add... Remove Edit...
Wed 10 Thu 11 Fri 12 Sat 13
Mary's Birthday
09:00 - 11:00 Important event
08:30 - 10:30 Buy birthday present for Mary
09:00 - 11:30 Go fishing
12:00 - 13:00 Lunch
Lunch
12:30 Tennis
13:30 - 16:30 Meeting with the boss
14:15 - 16:15 Feed the penguins

Figure 7: *Kontact screenshot showing the active calendar component*

7 Availability

Kontact is Free Software. Its library and interface parts are available under the GNU Library General Public License (LGPL) [28], the application itself is available under the GNU General Public License (GPL) [27].

A first stable source code release of Kontact was done in March 2003 [16]. The latest stable version was released as part of KDE 3.2 in February 2004. It can be downloaded from the KDE FTP server [17]. The next stable release (Kontact 1.0) is planned for mid of 2004. This will be the first separate release of the kdepim module independent of the other KDE modules. It will be based on the KDE 3.2 libraries.

8 Conclusion

Kontact introduces a new level of desktop application integration based on the technologies of the KDE framework. It provides mail, organizer, contact and other components to deliver a solution for personal information management and groupware. Its first full-featured stable release is available since the beginning of 2004.

In addition to the integration on a technical level Kontact has also integrated the development communities forming a stronger and more productive community on a higher level. This nicely demonstrates the power of Free Software development.

9 About The Authors

This paper was written by Cornelius Schumacher. He is a long-term kdepim contributor, one of the founders of Kontact and maintainer of several KDE applications and libraries, one of them being KOrganizer. He acts as release coordinator for the Kontact 1.0 release.

The other founding authors of Kontact are Daniel Molkentin, maintainer of the Kontact framework application, and Don Sanders, one of the founders of kdepim and co-maintainer of KMail. The original framework code was written by Matthias Hoelzer-Kluepfel.

David Faure, Tobias Koenig, maintainer of KAddressBook and Ingo Kloecker, maintainer of KMail, have done important contributions to Kontact and in particular to the Kontact application integration framework.

But the most important contributors to Kontact are the developers, translators, documentation writers and other contributors of the applications embedded into the Kontact framework. Thanks to all of them.

References

[1] Kontact Homepage, `http://www.kontact.org`

[2] KDE Homepage, `http://www.kde.org`

[3] GNOME Homepage, `http://www.gnome.org`

[4] freedesktop.org, `http://www.freedesktop.org`

[5] David Sweet et. al., *KDE 2.0 Development*, SAMS (2000), `http://www.andamooka.org/kde20devel`.

[6] David Faure, *Coding with KParts*, `http://www-106.ibm.com/developerworks/library/l-kparts/`

[7] KParts API Documentation `http://api.kde.org/3.2-api/kparts/html/`

[8] Distributed Component Object Model (DCOM), `http://www.microsoft.com/com/tech/DCOM.asp`

[9] Common Object Request Broker Architecture (CORBA), `http://www.omg.org/gettingstarted/corbafaq.htm`

[10] Universal Network Objects (UNO), `http://udk.openoffice.org/`

[11] Bonobo document model, `http://developer.gnome.org/arch/component/bonobo.html`

[12] Inter-Client Exchange (ICE) Protocol, `http://www.xfree86.org/current/ice.html`

[13] D-BUS, `http://freedesktop.org/Software/dbus`

[14] Konqueror Homepage, `http://www.konqueror.org`

[15] KOffice Homepage, `http://www.koffice.org`

[16] Kontact 0.2.1, `http://kontact.org/download`

[17] KDE FTP Server, `http://ftp.kde.org`

[18] Post Office Protocol (POP), RFC 1939, `http://www.faqs.org/rfcs/rfc1939.html`

[19] Internet Message Access Protocol (IMAP), RFC 2060, `http://www.faqs.org/rfcs/rfc2060.html`

[20] Simple Mail Transfer Protocol (SMTP), RFC 821, `http://www.faqs.org/rfcs/rfc821.html`

[21] vCard 2.1 Specification, `http://www.imc.org/pdi/vcard-21.txt`

[22] vCard 3.0 Specification, RFC 2425, `http://www.imc.org/rfc2425`, RFC 2426, `http://www.imc.org/rfc2426`

[23] Lightweight Directory Access Protocol (LDAP), RFC 3377, `http://www.faqs.org/rfcs/rfc3377.html`

[24] Internet Calendaring and Scheduling Core Object Specification (iCalendar), RFC 2445, `http://www.imc.org/rfc2445`

[25] iCalendar Transport-Independent Interoperability Protocol (iTIP), RFC 2446, `http://www.imc.org/rfc2446`

[26] iCalendar Message-based Interoperability Protocol (iMIP), RFC 2447, `http://www.imc.org/rfc2447`

[27] GNU General Public Licence, `http://www.gnu.org/copyleft/gpl.html`

[28] GNU Library General Public Licence, `http://www.gnu.org/copyleft/lgpl.html`

[29] Qt Toolkit, `http://www.trolltech.com`

[30] Qt Designer, `http://www.trolltech.com`

[31] Kolab, `http://www.kolab.org`

[32] eGroupware, `http://www.egroupware.org`

[33] Bugzilla, `http://www.bugzilla.org`

[34] KDE Kontact Commercial Improvement System, `http://www.kontact.org/shopping`

[35] Desktop Entry Standard, `http://www.freedesktop.org/Standards/desktop-entry-spec`

[36] KDevelop, `http://www.kdevelop.org`