CDE Plug-and-Play

By George Kraft IV

The programming infrastructure, such as ToolTalk, is a major strength of the Common Desktop Environment. This article illustrates client and server plug-and-play through the use of the Desktop's Application Programming Interfaces (APIs).

ToolTalk, in the Common Desktop Environment (CDE), is a message brokering system that enables applications to communicate with each other without having direct knowledge of one another. Client and server applications can be developed independently, mixed and matched, and upgraded separately through plug-and-play. In addition, the Desktop Service can be called to perform methods on file and buffer objects on behalf of ToolTalk.

Figure 1 shows the ToolTalk Service listening for `TtChmod` client requests. ToolTalk Service brokers pattern-matched `Chmod` messages to the registered mock change-mode application server (`ttchmodd`) that is waiting to handle the incoming messages.

ToolTalk brokers the requests from the client to the server application. The Desktop Service can forward CDE object method invocations to the ToolTalk Service. With the Desktop, both C programs and `dtksh` scripts can initiate actions that are transmitted to the ToolTalk Service. Consequently, client invocations from the `dtaction` command line, Application Manager icons, and File Manager icons can be directed through the Desktop Service to ToolTalk application services. Therefore, double clicking on a file icon in the file manager can be plugged into a ToolTalk registered application by first routing through the Desktop action and data-type service.

Ptype

The key to the ToolTalk message brokering system is its ability to define process-type identifiers with specific operations and arguments. In Figure 2, the process-type (`ptype`) `TtChmod` will execute the ToolTalk change-mode daemon application `ttchmodd`. This occurs when the session operation `Chmod` with filename and mode arguments are matched from a request.

Compiling the `ptype` definition with the `tt_type_comp` utility will register services for ToolTalk client applications to call. Consider the `ptype` as a C header file describing an Application Programming Interface (API) and the compiled suite of ToolTalk Services definitions as a library of methods to call. For the list of installed process-type identifiers, try running `tt_type_comp -P` at the command line to dump the database to the screen.

Chmod Service

The file change-mode application (`ttchmod`) described in Figure 3 is simply the Motif® command widget. In Figure 4, the `ttchmodd` server application graphically prompts the user for a command, then calls the command callback `callCB` to execute it; however, for this example, the application just prints the command.

The file change-mode application quickly becomes a plug-and-play service when it registers itself with the ToolTalk Service, then listens for messages to be handled by its `ToolTalkCB` receiver routine.

George Kraft IV

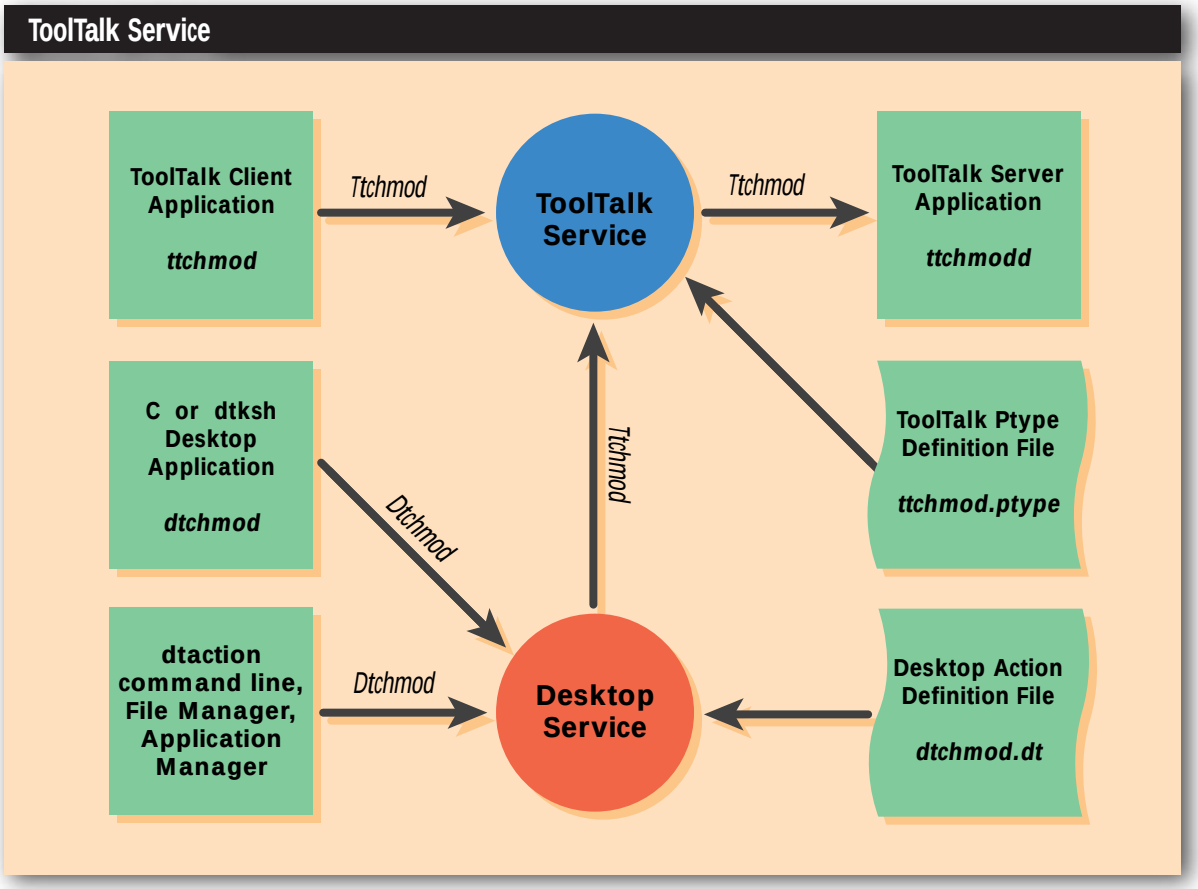

Figure 1. ToolTalk service

```
ptype TtChmod {
  start "/usr/tmp/ttchmodd";
  handle:
    session Chmod(
              in string filename,
              in string mode)
              => start opnum = 1;
};
```

Figure 2. TtChmod ptype definition

Register ToolTalk Service

An application must first locate the ToolTalk session associated with the X display to register itself as a service, as shown in Figure 5. After the application sets its default session to the display's session, the application can initiate itself as a ToolTalk process and obtain a ToolTalk file descriptor. When the `ttchmodd` application gets a handle on the ToolTalk session, then it can register the `TtChmod` process type and join the session to listen for requests.

Handle ToolTalk Requests

ToolTalk sends a message to a registered service; the service listens on its ToolTalk file descriptor for input. When input is observed, the `ToolTalkCB` routine is triggered where the message is read and analyzed. The message's operation is checked, then the arguments are read from the message. ToolTalk messages are similar to a reenterantive version of an ordinary C program's command-line arguments.

```
#include <stdio.h>
#include <Xm/Command.h>

#include <stdio.h>
#include <Tt/tt_c.h>

Widget          topLevel;
XtAppContext    appContext;
Display         *display;

void CmdCB         (Widget, XtPointer, XtPointer);
int  Register      ();
void ToolTalkCB    (Widget, XtPointer, XtPointer);
int  ToolTalkAbort (char *, Tt_status);

void main (argc, argv)
        int     argc;
        char    *argv[];
{
        Widget  cmd;
        int     ttfd;

        topLevel = XtVaAppInitialize(
                &appContext,
                "TtChmodd",
                NULL, 0,
                &argc, argv,
                NULL,
                NULL);

        cmd = XtVaCreateManagedWidget(
                "command",
                xmCommandWidgetClass,
                topLevel,
                NULL);

        XtAddCallback(cmd, XmNcommandEnteredCallback, CmdCB, NULL);

        ttfd = Register();

        XtAppAddInput(appContext, ttfd, (XtPointer)XtInputReadMask,
                ToolTalkCB, cmd);

        XtRealizeWidget(topLevel);

        XtAppMainLoop(appContext);

        tt_close();
}

void CmdCB (w, clientData, callData)
Widget          w;              /*  widget id           */
XtPointer       clientData;     /*  data from application   */
XtPointer       callData;       /*  data from widget class  */
{
        XmCommandCallbackStruct *cbs = (XmCommandCallbackStruct *) callData;
        char *cmd;

        XmStringGetLtoR(cbs->value, XmSTRING_DEFAULT_CHARSET, &cmd);

        printf("CmdCB(): %s\n", cmd);
}
```

Figure 3. Ttchmodd: command widget

The ttchmodd service is no longer needed after it reads the message; so the recipient tells the ToolTalk service to discard the message.

Figure 4. ttchmodd

ToolTalk Client

The ttchmod ToolTalk client is much simpler than the ttchmodd ToolTalk server. The client opens a ToolTalk process and locates the session. The TtChmod process-type message request consists of the Chmod operation with the filename and mode input arguments. It is sent to the ToolTalk Service to be brokered to a registered server application accepting the ptype pattern. However, note that this example omits error checking and garbage collection with tt_mark and tt_release.

Desktop Action

The Desktop Action database in CDE describes methods and objects for applications to act upon. CDE's Desktop Service can describe an action like DtChmod, shown in Figure 8, that can be forwarded through the ToolTalk Service to ttchmodd. If the action does not receive the appropriate arguments, then the Desktop can prompt the user, as shown in Figure 9.

The relationship of the Desktop Action definitions to CDE methods is similar to the

```
Register()
{
        int     ttfd;
        char    *session;
        char    *procid;
        Tt_status ttrc;
        int     ttmark;

        ttmark = tt_mark();

        session = tt_X_session(DisplayString(XtDisplay(topLevel)));
        ttrc = tt_default_session_set(session);
        ToolTalkAbort("TtServer(): tt_default_session_set", ttrc);

        procid = tt_open();
        ToolTalkAbort("TtServer(): tt_open", tt_ptr_error(procid));

        ttfd = tt_fd();
        ToolTalkAbort("TtServer(): tt_fd", tt_int_error(ttfd));

        ttrc = tt_ptype_declare("TtChmod");
        ToolTalkAbort("TtServer(): tt_ptype_declare", ttrc);

        ttrc = tt_session_join(tt_default_session());
        ToolTalkAbort("TtServer(): tt_session_join", ttrc);

        tt_release(ttmark);

        return(ttfd);
}
```

Figure 5. Ttchmodd: ToolTalk registration

```
void ToolTalkCB (w, clientData, callData)
Widget          w;              /*  widget id           */
XtPointer       clientData;     /*  data from application   */
XtPointer       callData;       /*  data from widget class  */
{
        Tt_message incoming;
        Tt_status ttrc;
        int ttmark, args;
        char *filename;
        char *mode;
        char command[BUFSIZ];
        XmString xmcmd;

        ttmark = tt_mark();

        incoming = tt_message_receive();
        ToolTalkAbort("ToolTalkCB(): tt_message_receive",
                tt_ptr_error(incoming));

        if (incoming && (0 == strcmp(tt_message_op(incoming), "Chmod"))) {

                args = tt_message_args_count(incoming);
                ToolTalkAbort("ToolTalkCB(): tt_message_args_count",
                        tt_int_error(args));

                filename = tt_message_arg_val(incoming, 0);
                ToolTalkAbort("ToolTalkCB(): tt_message_arg_val",
                        tt_ptr_error(filename));

                mode = tt_message_arg_val(incoming, 1);
                ToolTalkAbort("ToolTalkCB(): tt_message_arg_val",
                        tt_ptr_error(mode));

                ttrc = tt_message_reply(incoming);
                ToolTalkAbort("ToolTalkCB(): tt_message_destroy", ttrc);

                sprintf(command, "/bin/chmod %s %s\n", mode, filename);
                xmcmd = XmStringCreateLocalized(command);
                XmCommandSetValue(w, xmcmd);
                XmStringFree(xmcmd);
        } else {
                printf("ToolTalkCB(): unknown message %s.\n",
                        tt_message_op(incoming));
        }

        ttrc = tt_message_destroy(incoming);
        ToolTalkAbort("ToolTalkCB(): tt_message_destroy", ttrc);
        tt_release(ttmark);
}

ToolTalkAbort(char *procname, Tt_status errid)
{
        if (tt_is_err(errid)) {
                fprintf(stderr, "%s returned ToolTalk error: %s\n",
                        procname, tt_status_message(errid));
                exit(1);
        }
}
```

Figure 6. Ttchmodd: ToolTalk message receiver

```
#include <stdio.h>
#include <Tt/tt_c.h>

main(int argc, char *argv[])
{
        Tt_message msg;
        Tt_status ttstat;

        if (argc != 3) {
                printf("Usage: %s <mode> <filename>\n", argv[0]);
                exit(1);
        }

        tt_open();
        msg = tt_message_create();
        tt_message_scope_set(msg, TT_SESSION);
        tt_message_session_set(msg, tt_default_session());
        tt_message_class_set(msg, TT_REQUEST);
        tt_message_handler_set(msg, "TtChmod");
        tt_message_address_set(msg, TT_PROCEDURE);
        tt_message_op_set(msg, "Chmod");
        tt_message_arg_add(msg, TT_IN, "string", argv[2]);
        tt_message_arg_add(msg, TT_IN, "string", argv[1]);
        ttstat = tt_message_send(msg);
        if (ttstat != TT_OK) {
                printf("%s\n", tt_status_message((int) ttstat));
                exit(1);
        }
}
```

Figure 7. Ttchmod: ToolTalk client

```
ACTION DtChmod
{
        LABEL           Chmod
        ICON            DtDFile
        TYPE            TT_MSG
        TT_CLASS        TT_REQUEST
        TT_SCOPE        TT_SESSION
        TT_ARG0_MODE    TT_IN
        TT_ARG0_VTYPE   string
        TT_ARG0_VALUE   %Arg_1"File name:"%
        TT_ARG1_MODE    TT_IN
        TT_ARG1_VTYPE   string
        TT_ARG1_VALUE   %Arg_2"Octel mode:"%
        TT_OPERATION    Chmod
}
```

Figure 8. DtChmod CDE action definition

relationship of `ptype` definitions to ToolTalk processes. For an example of Desktop actions and data types, run `dttypes` as the command line to dump the database to the screen.

Desktop Service Client

The APIs of the Desktop Service can invoke actions registered in the Desktop database either from a C program or from a `dtksh` script, as shown in Figure 10. Here, the `dtchmod.ds`, `dtksh` script prompts the user, as shown in Figure 11, with the message dialogue to confirm with the user before requesting changes to the file's mode.

Command Line

In addition to calling Desktop actions from C programs and `dtksh` shell scripts, users can initiate requests from the command line, as shown in Figure 12. If the appropriate arguments are given, then the action is forwarded to ToolTalk; otherwise the user is first queried, as shown in Figure 9.

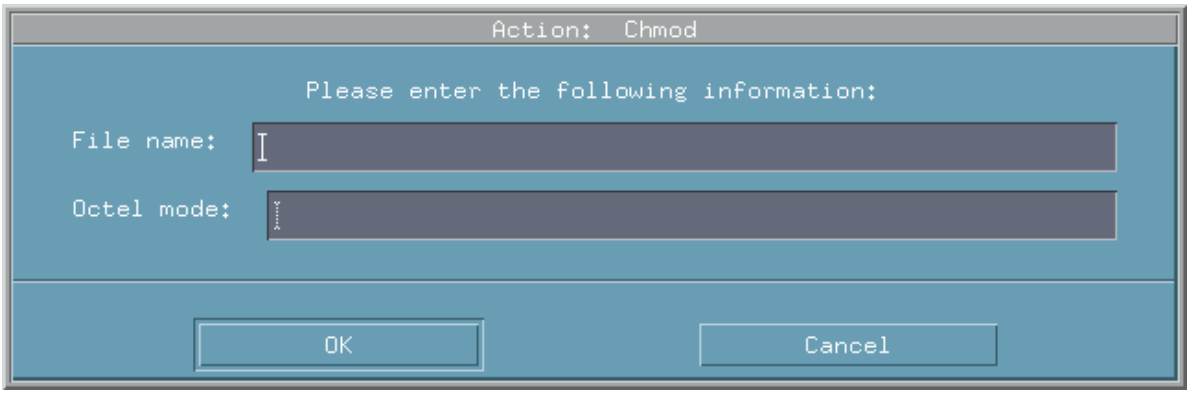

Figure 9. dtaction

```
#!/usr/dt/bin/dtksh

chmodCB()
{
        DtActionInvoke $TOPLEVEL DtChmod '' '' '' True NULL \
                "FILE" "$FILE" \
                "FILE" "$MODE"
}

main()
{
        XtInitialize TOPLEVEL dtChmod DtChmod "$@"

        DtDbLoad

        XmCreateMessageDialog CHMOD $TOPLEVEL motd \
                dialogTitle:"DtChmod" \
                helpLabelString:"Chmod" \
                messageString:"$(print Chmod $FILE to $MODE)"

        XmMessageBoxGetChild OK $CHMOD DIALOG_OK_BUTTON
        XtUnmanageChild $OK

        XtAddCallback $CHMOD helpCallback chmodCB
        XtAddCallback $CHMOD cancelCallback exit

        XtManageChild $CHMOD

        XtMainLoop
}

MODE=$1
FILE=$2

main
```

Figure 10. DtChmod command-line action

Figure 11. dtchmod

Plug, and Play...

We have seen how the `ttchmodd` service registered with ToolTalk can receive messages matching the `TtChmod ptype` pattern from ToolTalk clients, from Desktop clients written in either C or `dtksh`, from the command line, and from double clicking on file object icons. Understanding these examples demonstrates how client and server applications can be developed independently, mixed and matched, and upgraded separately through plug-and-play. A ToolTalk-enabled application service registered with its `ptype` definition can be developed without specific knowledge of its counterpart.

CDE defines a message dictionary of Desktop-specific ToolTalk process types, operations, and

```
dtaction DtChmod /etc/motd 644
```

Figure 12. DtChmod command-line action

arguments as seen from viewing the database. Others, such as Computer-Aided Design (CAD) and Electronic Design Automation (EDA) services have developed supplemental dictionaries. You can use existing `ptypes` or define your own, but the important point to know is how to register the process-type identifier, operation, and arguments.

For more information regarding the Common Desktop Environment, visit IBM's CDE Web page at URL http://www.austin.ibm.com/software/CDE/.

George Kraft IV, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Kraft is an advisory programmer for the RS/6000 Division working in the AIX Desktop group. He is currently working on the Common Desktop Environment Version 2.1 and a version of the Network Computer. Mr. Kraft received a BS in Computer Science and Mathematics from Purdue University.*