RPC-Codegenerierung mit Perl

Martin Vorländer
PDV-SYSTEME GmbH

Die Ausgangslage

- **MEDAS**
  - **Messdaten-Erfassungs- und -Archivierungs-System**
  - **für Industrie-Prozesse; minimale Auflösung: 1 Sekunde**
  - **Ankopplung an Prozeßleitsysteme u.a. von ABB und Siemens**
  - **läuft unter OpenVMS**
  - **überwiegend in PASCAL geschrieben**
  - **wird seit 15 Jahren ständig weiterentwickelt**
  - **Zugriff bisher über VT-Terminals (auch VT340 - Grafik!) und DECwindows/Motif**
  - **eine rudimentäre, netzwerkfähige API existiert**

Die Aufgabenstellung

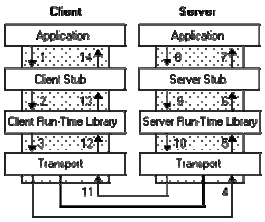
- eine neue API wurde entworfen
  - 330 Funktionen
  - Implementation in PASCAL unter OpenVMS
  - nicht netzwerkfähig
- neue Clients werden mit C++ unter Windows NT/2000/XP entwickelt
- Anbindung an das MEDAS-System über TCP/IP
- Die API muß netzwerkfähig gemacht werden und eine Win32-C(++)-Implementation bekommen

Aufgaben einer Netzwerk-API

- Verbergen der Plattform-Unterschiede zwischen Client und Server
  - Little-/Big-Endian
  - Unterschiede in Datentypen, z.B. int, float
- Übertragung von strukturierten Typen
- Der Aufruf einer Prozedur sollte wie ein lokaler Aufruf funktionieren

RPC

- Ursprünglich von SUN für sein NFS entwickelt
- Remote Procedure Call
  - Parameter auf dem Client „einpacken“ d.h. ins Netzwerkformat konvertieren
  - Aufruf & Parameter übertragen
  - Parameter auf dem Server „auspacken“ d.h. ins server-spezifische Format konvertieren
  - API-Funktion aufrufen
  - Ergebnisse „einpacken“
  - Ergebnisse übertragen
  - Ergebnisse auf dem Client „auspacken“ d.h. ins client-spezifische Format konvertieren



RPC: Implementierungen

- zwei (inkompatible) Varianten:
  - Distributed Computing Environment (DCE) RPC
    - *beim Win32 TCP/IP Stack enthalten*
    - *Runtime-Paket in der OpenVMS-Lizenz enthalten*
    - *Application Developers Kit kommerziell (QL-24GA*-AA)*
  - Open Network Computing (ONC) RPC
    - *SUNs RPC*
    - *bei den OpenVMS TCP/IP Stacks enthalten*
    - *für Win32 sowohl kommerzielle als auch OpenSource Implementationen*
- Entscheidung zugunsten von ONC RPC

ONC RPC

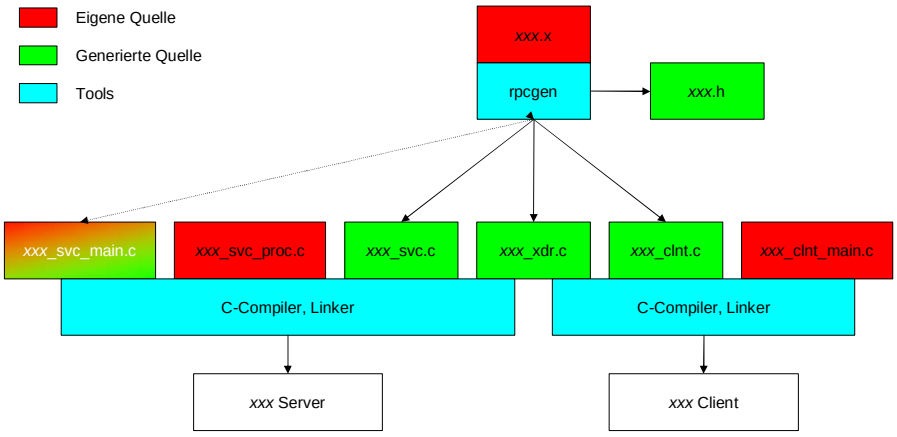
- RFC 1057, RFC 1831
- eine Prozedur wird spezifiziert durch
  - IP-Adresse oder Hostname
  - Transport: TCP oder UDP
  - Programm-Nummer und -Version
  - Prozedur-Nummer
- Ein- und Ausgabe-Variablen werden in zwei Strukturen spezifiziert

XDR

- External Data Representation
- RFC 1014, RFC 1832
- Übersetzung von Datentypen von und in eine Netzwerk-Repräsentation
- Basis sind Big-Endian-Langworte (4 Byte)
- Verfügbare Typen:

Boolean	(Fixed-length|Variable-length) Array
[Unsigned] [Hyper] Integer	Structure
[Double-precision|Quadruple-precision] Floating-point	Discriminated Union
Enumeration	(Fixed-length|Variable-length) Opaque Data
String	Void

RPC: Ein- und Ausgaben



An die Arbeit!

- alle Funktions-Deklarationen liegen in PASCAL-Header-Dateien vor
- Idee: Parsen der Deklarationen und Generieren von RPC- und API-Quellen
- Vorteil: Da die API zum Zeitpunkt des Arbeitsbeginns noch nicht vollständig definiert war, gäbe es bei Änderungen an der API keine Nacharbeit
- Perl bietet mit Parse::RecDescent einen kompletten „Recursive Descent“ Parser
- Perl ist unter OpenVMS, Windows, Linux,... vorhanden

Beispiel einer Funktion

```
CONST
  ChannelListLength = 3072;          {aus SRV:INTERFACE_DATA.INC}
TYPE
  TypeReturnCode = INTEGER;          {aus SRV:INTERFACE.INC}
  TypeChannelList =                  {aus SRV:INTERFACE_DATA.INC}
    PACKED ARRAY [1..ChannelListLength] OF CHAR;
  TypStatusByte = PACKED ARRAY [1..8] OF 0..1;        {aus GLB:TYPSTS.INC}
  TypStatusRec = RECORD              {aus GLB:TYPSTS.INC}
    Hardware_Status : TypStatusByte;
    MEDAS_Status    : TypStatusByte;
    Alarm_Status    : TypStatusByte;
    Aux_Status      : TypStatusByte;
  END;
  TypZeit = UNSIGNED;                {aus GLB:TYPTIM.INC}
  TypeValueRecord = RECORD           {aus SRV:INTERFACE_DATA.INC}
    Value     : REAL;
    Status    : TypStatusRec;
    TimeStamp : TypZeit;
  END;

FUNCTION ReadMultipleValues          {aus SRV:INTERFACE_DATA.INC}
 ( VAR ChannelList  : TypeChannelList;                          {I}
   VAR ValueCount   : INTEGER;                                  {O}
   VAR ChannelArray : ARRAY [j0..j1:INTEGER] OF INTEGER;        {O}
   VAR ValueArray   : ARRAY [i0..i1:INTEGER] OF TypeValueRecord {O}
  ) : TypeReturnCode; EXTERNAL;
```


m2srv.x (1/2)

```
const ChannelListLength = 3072;

typedef int TypeReturnCode;

typedef u_char TypeChannelList[ChannelListLength];

typedef u_char TypStatusByte[1];

struct TypStatusRec {
  TypStatusByte Hardware_Status;
  TypStatusByte MEDAS_Status;
  TypStatusByte Alarm_Status;
  TypStatusByte Aux_Status;
};

typedef u_int TypZeit;

struct TypeValueRecord {
  float Value;
  TypStatusRec Status;
  TypZeit TimeStamp;
};
```

m2srv.x (2/2)

```
struct arg_ReadMultipleValues_t {
        TypeChannelList ChannelList;
        int ChannelArray<>;
        TypeValueRecord ValueArray<>;
};

struct res_ReadMultipleValues_t {
        TypeReturnCode _retCode;
        int ValueCount;
        int ChannelArray<>;
        TypeValueRecord ValueArray<>;
};

program M2SRV_PROG {
  version M2SRV_VERSION {
    ...
    res_ReadMultipleValues_t ReadMultipleValues(arg_ReadMultipleValues_t) = 141;
    ...
  } = 1;
} = 0x24242424;
```


m2srv.h (1/3)

```
#define ChannelListLength 3072

typedef int TypeReturnCode;
bool_t xdr_TypeReturnCode();

typedef u_char TypeChannelList[ChannelListLength];
bool_t xdr_TypeChannelList();

typedef u_char TypStatusByte[1];
bool_t xdr_TypStatusByte();

struct TypStatusRec {
  TypStatusByte Hardware_Status;
  TypStatusByte MEDAS_Status;
  TypStatusByte Alarm_Status;
  TypStatusByte Aux_Status;
};
typedef struct TypStatusRec TypStatusRec;
bool_t xdr_TypStatusRec();

typedef u_int TypZeit;
bool_t xdr_TypZeit();
```

m2srv.h (2/3)

```
struct TypeValueRecord {
  float Value;
  TypStatusRec Status;
  TypZeit TimeStamp;
};
typedef struct TypeValueRecord TypeValueRecord;
bool_t xdr_TypeValueRecord();

struct res_ReadMultipleValues_t {
  TypeReturnCode _retCode;
  int ValueCount;
  struct {
    u_int ChannelArray_len;
    int *ChannelArray_val;
  } ChannelArray;
  struct {
    u_int ValueArray_len;
    TypeValueRecord *ValueArray_val;
  } ValueArray;
};
typedef struct res_ReadMultipleValues_t res_ReadMultipleValues_t;
bool_t xdr_res_ReadMultipleValues_t();
```


m2srv.h (3/3)

```
struct arg_ReadMultipleValues_t {
  TypeChannelList ChannelList;
  struct {
    u_int ChannelArray_len;
    int *ChannelArray_val;
  } ChannelArray;
  struct {
    u_int ValueArray_len;
    TypeValueRecord *ValueArray_val;
  } ValueArray;
};
typedef struct arg_ReadMultipleValues_t arg_ReadMultipleValues_t;
bool_t xdr_arg_ReadMultipleValues_t();

#define M2SRV_PROG 0x24242424
#define M2SRV_VERSION 1

#define ReadMultipleValues ((u_long)141)
extern res_ReadMultipleValues_t *readmultiplevalues_1();
```

m2srv_xdr.c

```
bool_t
xdr_res_ReadMultipleValues_t(xdrs, objp)
XDR *xdrs;
res_ReadMultipleValues_t *objp;
{
  if (!xdr_TypeReturnCode(xdrs, &objp->_retCode)) {
    return (FALSE);
  }
  if (!xdr_int(xdrs, &objp->ValueCount)) {
    return (FALSE);
  }
  if (!xdr_array(xdrs, (char **)&objp->ChannelArray.ChannelArray_val,
                 (u_int *)&objp->ChannelArray.ChannelArray_len, ~0,
                 sizeof(int), xdr_int)) {
    return (FALSE);
  }
  if (!xdr_array(xdrs, (char **)&objp->ValueArray.ValueArray_val,
                 (u_int *)&objp->ValueArray.ValueArray_len, ~0,
                 sizeof(TypeValueRecord), xdr_TypeValueRecord)) {
    return (FALSE);
  }
  return (TRUE);
}
```


m2srv_svc.c (1/2)

```
void
m2srv_prog_1(rqstp, transp)
struct svc_req *rqstp;
register SVCXPRT *transp;
{
  union {
    ...
    arg_ReadMultipleValues_t readmultiplevalues_1_arg;
    ...
  } argument;
  char *result;
  xdrproc_t _xdr_argument, _xdr_result;
  char *(*local)(char *, struct svc_req *);

  switch (rqstp->rq_proc) {
    ...
    case ReadMultipleValues:
      _xdr_argument = (xdrproc_t) xdr_arg_ReadMultipleValues_t;
      _xdr_result = (xdrproc_t) xdr_res_ReadMultipleValues_t;
      local = (char *(*)(char *, struct svc_req *)) readmultiplevalues_1_svc;
      break;
    ...
  }
```

m2srv_svc.c (2/2)

```
  memset ((char *)&argument, 0, sizeof (argument));
  if (!svc_getargs (transp, _xdr_argument, (caddr_t) &argument)) {
    svcerr_decode (transp);
    return;
  }

  result = (*local)((char *)&argument, rqstp);

  if (result != NULL && !svc_sendreply(transp, _xdr_result, result)) {
    svcerr_systemerr (transp);
  }

  if (!svc_freeargs (transp, _xdr_argument, (caddr_t) &argument)) {
    fprintf (stderr, "%s", "unable to free arguments");
    exit (1);
  }

  return;
}
```


m2srv_svc_main.c

```
main (argc, argv)
int argc;
char **argv;
{
  register SVCXPRT *transp;

  pmap_unset (M2SRV_PROG, M2SRV_VERSION);

  transp = svctcp_create(RPC_ANYSOCK, 0, 0);
  if (transp == NULL) {
    fprintf (stderr, "%s", "cannot create tcp service.");
    exit(1);
  }
  if (!svc_register(transp, M2SRV_PROG, M2SRV_VERSION,
                    m2srv_prog_1, IPPROTO_TCP)) {
    fprintf (stderr, "%s",
             "unable to register (M2SRV_PROG, M2SRV_VERSION, tcp).");
    exit(1);
  }

  svc_run ();
  fprintf (stderr, "%s", "svc_run returned");
  exit (1);
  /* NOTREACHED */
}
```

m2srv_clnt.c

```
res_ReadMultipleValues_t *
readmultiplevalues_1(arg_ReadMultipleValues_t *argp, CLIENT *clnt)
{
  static res_ReadMultipleValues_t clnt_res;

  memset((char *)&clnt_res, 0, sizeof(clnt_res));
  if (clnt_call (clnt, ReadMultipleValues,
                 (xdrproc_t) xdr_arg_ReadMultipleValues_t, (caddr_t) argp,
                 (xdrproc_t) xdr_res_ReadMultipleValues_t, (caddr_t) &clnt_res,
                 TIMEOUT) != RPC_SUCCESS) {
    return (NULL);
  }
  return (&clnt_res);
}
```


Parse::RecDescent

- „Recursive Descent“ Parser
- geschrieben von Prof. Damian Conway
- arbeitet mit BNF-ähnlichen Grammatik-Regeln
- generiert aus Grammatik-Regeln Perl-Code
- objektorientiert
  - je Regel eine Klasse

Eine Grammatik-Regel

```
routine_declaration:
          /FUNCTION/i  <commit> identifier parameter_list(?) ':' result_type_id
                {
                        $return =
                        $::routine_decl{lc $item{identifier}{__VALUE__}} =
                            bless \%item, $item[0];
                        $return->{__file__} = $::current_file;
                }
        | /PROCEDURE/i <commit> identifier parameter_list(?)
                {
                        $return =
                        $::routine_decl{lc $item{identifier}{__VALUE__}} =
                            bless \%item, $item[0];
                        $return->{__file__} = $::current_file;
                }
        | <error?>
```


Die Grammatik

- insgesamt 65 Regeln
- keine komplette PASCAL-Grammatik
- genug für die PASCAL-Include-Dateien
- genauer: fast genug für PASCAL-Module
- keine Statements
- keine Schemata außer Conformant-Array-Parametern

Die Struktur-Verarbeitung

- Parsen und Verarbeiten der Strukturen gegenüber der Codegenerierung die weitaus aufwendigere Aufgabe
- daher Zweiteilung
- Nach dem Parser-Lauf:
  - Identifizieren der benötigten Funktionen anhand einer Liste
  - Identifizieren aller benötigten Typen und Konstanten
  - Reduzieren der Konstanten-, Typen- und Funktions-Hashes
  - Speichern der Hashes mit dem Storable-Modul

Die Codegenerierung

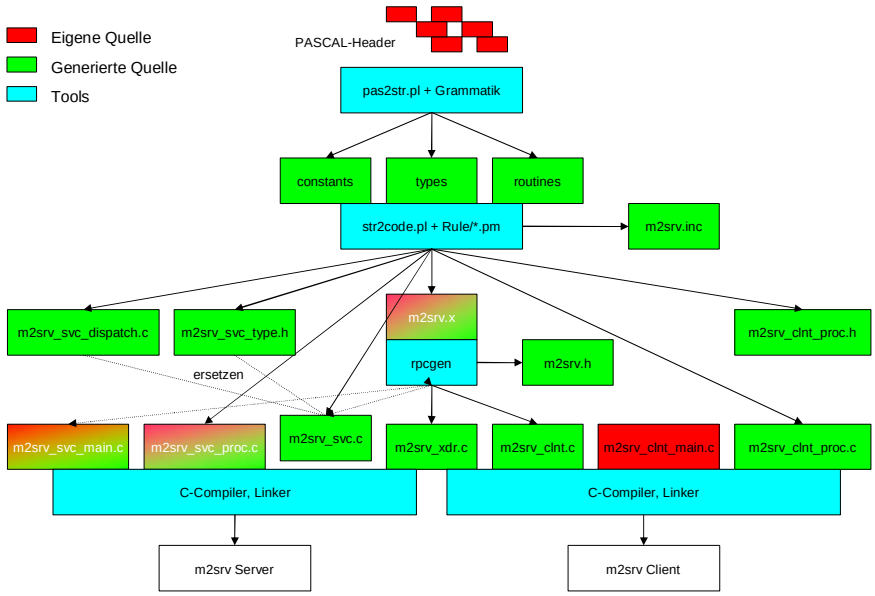
- Schnittstelle zwischen API und RPC
- neue Methoden in den Klassen, z.B. (Auszug):

```
package Rule::routine_declaration;

use Rule::identifier;
use Rule::parameter_list;
use Rule::result_type_id;

sub pascal_inc {
       my $self = shift;
       my $ret =
         $self->{__PATTERN1__} # /FUNCTION|PROCEDURE/
       . ' '
       . $self->{identifier}->value;
       $ret .= $self->{parameter_list}[0]->pascal_inc
               if @{$self->{parameter_list}};
       $ret .= ' : ' . $self->{result_type_id}->pascal_inc
               if exists $self->{result_type_id};
       return $ret;
}
```

Ein- und Ausgaben II



m2srv_svc_proc.c (1/2)

```
#undef ReadMultipleValues

res_ReadMultipleValues_t *
readmultiplevalues_1_svc(
  arg_ReadMultipleValues_t * clnt_arg,
  struct svc_req * rqstp
) {
  static res_ReadMultipleValues_t clnt_res;

  TypeChannelList ChannelList;
  CONFORMANT_ARRAY_STRUCT(j0j1, int, ChannelArray);
  CONFORMANT_ARRAY_STRUCT(i0i1, TypeValueRecord, ValueArray);
  TypeReturnCode _retCode;
  int ValueCount;

  /* Copy received data from RPC struct */
  memcpy(ChannelList, clnt_arg->ChannelList, sizeof(TypeChannelList));

  nca_ChannelArray.dsc.dsc$w_length  = sizeof(int);
  nca_ChannelArray.dsc.dsc$b_dtype   = DSC$K_DTYPE_L;
  nca_ChannelArray.dsc.dsc$b_class   = DSC$K_CLASS_NCA;
  nca_ChannelArray.dsc.dsc$a_pointer =
   (char *)clnt_arg->ChannelArray.ChannelArray_val;
  ...
```

m2srv_svc_proc.c (2/2)

```
  nca_ValueArray.dsc.dsc$w_length  = sizeof(TypeValueRecord);
  nca_ValueArray.dsc.dsc$b_dtype   = DSC$K_DTYPE_Z;
  nca_ValueArray.dsc.dsc$b_class   = DSC$K_CLASS_NCA;
  nca_ValueArray.dsc.dsc$a_pointer =
    (char *)clnt_arg->ValueArray.ValueArray_val;
  ...

  _retCode = ReadMultipleValues(
    &ChannelList,
    &ValueCount,
    &nca_ChannelArray,
    &nca_ValueArray
  );

  /* Copy data to send to RPC struct */
  memcpy(&clnt_res._retCode, &_retCode, sizeof(TypeReturnCode));
  memcpy(&clnt_res.ValueCount, &ValueCount, sizeof(int));
  clnt_res.ChannelArray.ChannelArray_len =
    nca_ValueArray.dsc$bounds[0].dsc$l_u + 1;
  clnt_res.ChannelArray.ChannelArray_val =
    (int *)nca_ChannelArray.dsc.dsc$a_pointer;
  clnt_res.ValueArray.ValueArray_len =
    nca_ValueArray.dsc$bounds[0].dsc$l_u + 1;
  clnt_res.ValueArray.ValueArray_val =
    (TypeValueRecord *)nca_ValueArray.dsc.dsc$a_pointer;
  return &clnt_res;
}
```


m2srv_clnt_proc.c (1/2)

```
CLIENT * client;

#undef ReadMultipleValues

TypeReturnCode ReadMultipleValues(
  /*R*/ TypeChannelList * ChannelList,
  /*W*/ int * ValueCount,
  /*M*/ CONFORMANT_ARRAY_PARAMETER(j0j1, int, ChannelArray),
  /*M*/ CONFORMANT_ARRAY_PARAMETER(i0i1, TypeValueRecord, ValueArray)
) {
  arg_ReadMultipleValues_t clnt_arg;
  res_ReadMultipleValues_t * clnt_res;
  TypeReturnCode _retCode;

  if (client == NULL) {
    return FAILURE_RETCODE;
  }

  /* Copy data to send to RPC struct */
  memcpy(clnt_arg.ChannelList, *ChannelList, sizeof(TypeChannelList));
  clnt_arg.ChannelArray.ChannelArray_len = ChannelArray_len;
  clnt_arg.ChannelArray.ChannelArray_val = ChannelArray_val;
  clnt_arg.ValueArray.ValueArray_len = ValueArray_len;
  clnt_arg.ValueArray.ValueArray_val = ValueArray_val;
```

m2srv_clnt_proc.c (2/2)

```
  clnt_res = readmultiplevalues_1((void*)&clnt_arg, client);
  if (clnt_res == (res_ReadMultipleValues_t *) NULL) {
    clnt_perror(client, "ReadMultipleValues");
    return FAILURE_RETCODE;
  }

  /* Copy received data from RPC struct */
  memcpy(&_retCode, &clnt_res->_retCode, sizeof(TypeReturnCode));
  memcpy(ValueCount, &(clnt_res->ValueCount), sizeof(int));
  memcpy(ChannelArray_val, clnt_res->ChannelArray.ChannelArray_val,
         clnt_res->ChannelArray.ChannelArray_len * sizeof(int));
  memcpy(ValueArray_val, clnt_res->ValueArray.ValueArray_val,
        clnt_res->ValueArray.ValueArray_len * sizeof(TypeValueRecord));

  return _retCode;
}
```


Stolpersteine (1/3)

- %INCLUDE**-Statement - kann wie ein Kommentar überall stehen**
  - **Behoben durch Änderung der PASCAL-Quellen**
  - %INCLUDE**-Statements nur noch auf Ebene von Deklarationen**
- **Informationstragende Kommentare bei Parametern (**{I}, {O}, {M}**)**
  - **Gewürdigt durch spezielle** <skip>**-Regel in der** parameter_list**-Regel**
- **Bitarrays:** PACKED ARRAY [*index*] OF 0..1
  - **Spezialfall ausprogrammiert**

Stolpersteine (2/3)

- SET OF *enum-type*
  - **Umgesetzt als** char[] **plus** enum**, der Masken enthält**
- **Conformant Array-Parameter:**
  ARRAY [i0..i1 : INTEGER] OF *type*
  - **RPC-Typ "variable length array"**
  - **wegen Problemen mit der TCPware-RPC-Implementation verworfen**
  - **statt dessen: Änderung der API auf feste Arrays**
- **Maximale Länge von Identifiern beim VMS-Linker: 31 Zeichen**
  - **Verkürzung der Identifier auf 21 Zeichen,**
    z.B. GetPCSConnectionCharacteristics **nach**
    GetPCSConnectionCha_1**,**
    so daß xdr_arg_getpcsconnectioncha_1_t **im Limit bleibt**

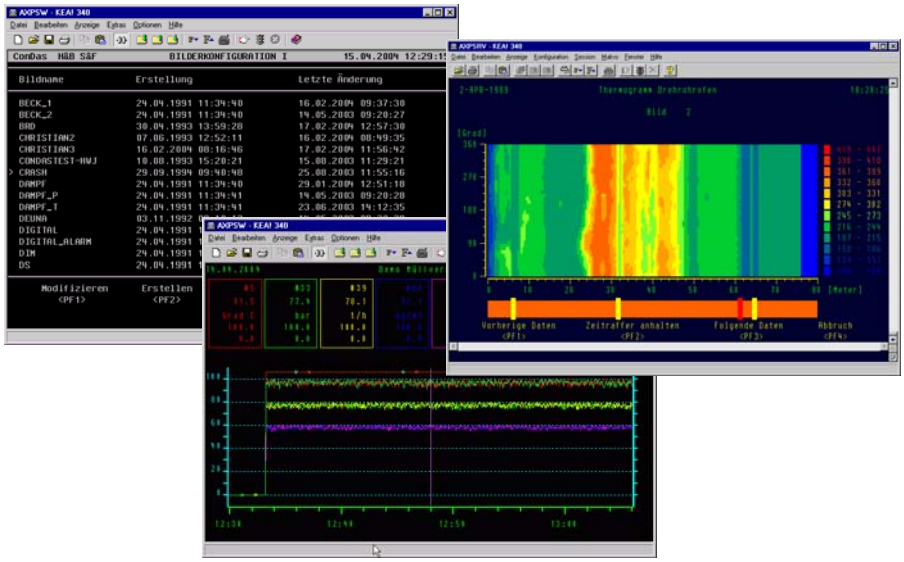
Stolpersteine (3/3)

- **Bedienung von mehreren Clients durch den TCPware RPC-Server**
  - **kein Multi-Threading**
  - **behoben durch Benutzung der (alten) asynchronen RPC-Schnittstelle (ASTs)**
- **MEDAS-Benutzer-Identifizierung durch PID**
  - **Trennung von Netzwerk- und MEDAS-Teil im Server**
  - **pro Verbindung ein MEDAS-Worker-Prozess**
  - **Datenübergabe über Global Sections**
- **Verbindungshandle „hinter den Kulissen“ unpraktikabel**
  - **Änderung der Client-API**
  - **Handle als zusätzlicher Parameter**

Ergebnisse

- **Eingabe:**
  - **97 aus 320 PASCAL-Include-Dateien**
  - **974 Konstanten**
  - **542 Typen**
  - **744 Routinen**
- **Ausgabe:**
  - **252 Konstanten**
  - **245 Typen**
  - **330 Routinen**

Vorher...



...und Nachher