RPC Code Generation with Perl

Martin Vorländer
PDV-SYSTEME GmbH

The Situation

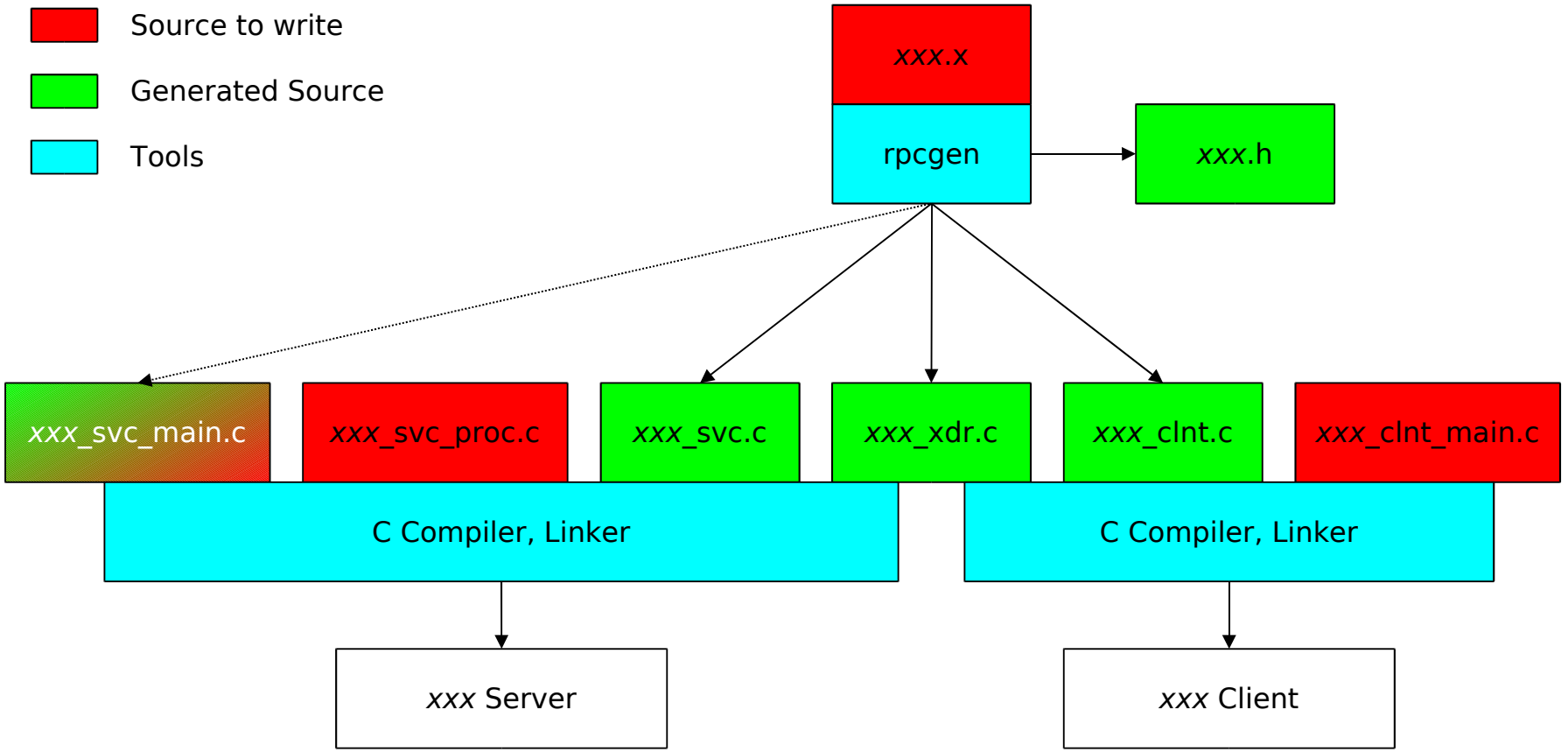
- **MEDAS**
  - **Measurement Aquirement and Archiving System**
  - **for industrial processes; minimal resolution: 1 second**
  - **runs under OpenVMS**

  - **Mainly written in PASCAL**
  - **Prior access via VT terminals (using VT340, even graphics – with 9600 bps!) and DECwindows/Motif**
  - **An old network API and Win32 clients exists**

The Task

- A new API is designed
- 330 functions
  - Implementation in PASCAL under OpenVMS
  - not networked
- New Clients are developed using C++ under Windows NT/2000/XP
- Communication with MEDAS via TCP/IP
- The API has to be made networkable with a Win32 C(++) implementation
- RPC

RPC: Input and Output



Ready, Steady, GO!

- **Idea: Parse the (constant, type, routine) declarations from the PASCAL header files and generate RPC and API C code**
- **Advantage: as the API wasn't fully defined when I started to work on this, there wouldn't be any work (for me, at least) when it changed**
- **Perl's Parse::RecDescent is a complete Recursive Descent Parser**
- **Perl implementations exist on OpenVMS, Windows, Linux,...**

A sample function

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

FUNCTION ReadMultipleValues          {from 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
- Written by Damian Conway
- uses grammar rules similar to Backus-Naur
- generates Perl code from these rules
- Object oriented
  - One class (=module) per grammar rule

A sample grammar rule

```
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?>
```

The Grammar

- 65 rules
- no complete PASCAL grammar
- Just enough for PASCAL include files
- more precisely: nearly enough for PASCAL modules
- No statements
- no schema types except Conformant Array parameters

The Code Generation

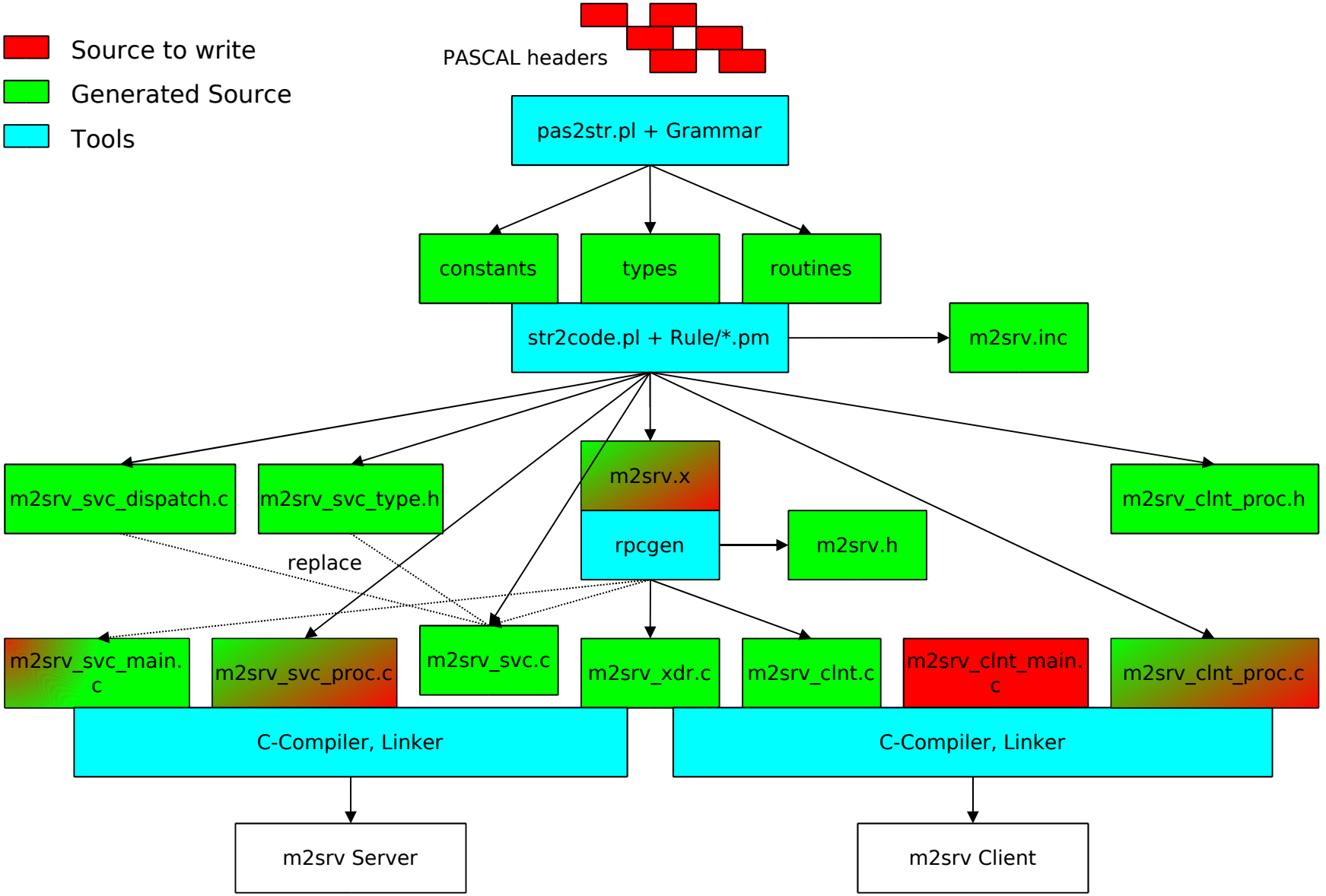
- **Interface between the API and RPC**
- **new class methods, e.g. (excerpt):**

```
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;
}
```

Input and Output 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;
}
```

Traps and Pitfalls (1/3)

- %INCLUDE **directives – like a comment, can be anywhere**
  - **fixed by changing the PASCAL source:**
  - %INCLUDE **statements only at declaration level**
- **Information carrying parameter comments ({I}, {O}, {M})**
  - **special** <skip> **rule in the** parameter_list **rule**
- **Bit arrays:** PACKED ARRAY [*index*] OF 0..1
  - **Special case, explicitly programmed out**

Traps and Pitfalls (2/3)

- SET OF *enum-type*
  - **Changed to** char[] **plus** enum **containing masks**
- **Conformant Array Parameter:**
  ARRAY [i0..i1 : INTEGER] OF *type*
  - **RPC type "variable length array"**
  - **Not working due to problems with the particular VMS TCP/IP RPC implementation**
  - **instead: Change API to use fixed length arrays**
- **Maximum identifier length with the VMS linker: 31 characters**
  - **Hash the identifiers to 21 characters, e.g.** GetPCSConnectionCharacteristics **to** GetPCSConnectionCha_1**, so that** xdr_arg_getpcsconnectioncha_1_t **is ≤ 31 char**

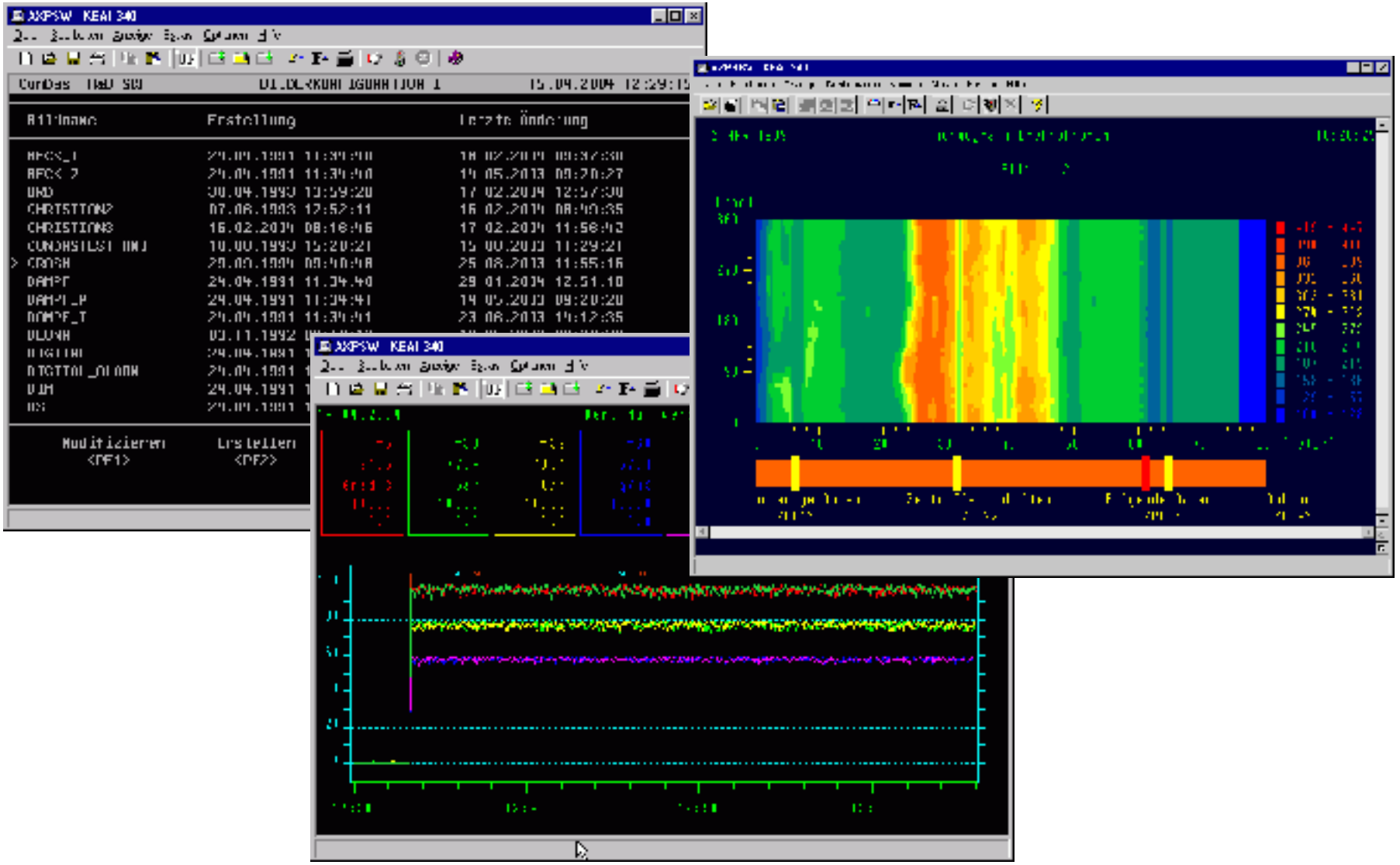
Traps and Pitfalls (3/3)

- MEDAS user identification by PID
  - Separation of Network and MEDAS part of the server
  - one MEDAS worker process per connection
  - Data transfer via shared memory
- Connection handle „behind the curtain“ is not practical
  - change the client API
  - Add handle as an additional parameter

Results

- **Input:**
  - **97 (out of 320) PASCAL include files**
  - **974 constants**
  - **542 types**
  - **744 routines**
- **Output:**
  - **252 constants**
  - **245 types**
  - **330 routines**

Before...



...and After