



RPC-Codegenerierung mit Perl

Martin Vorländer
PDV-SYSTEME GmbH

Die Ausgangslage

q MEDAS

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

Die Aufgabenstellung

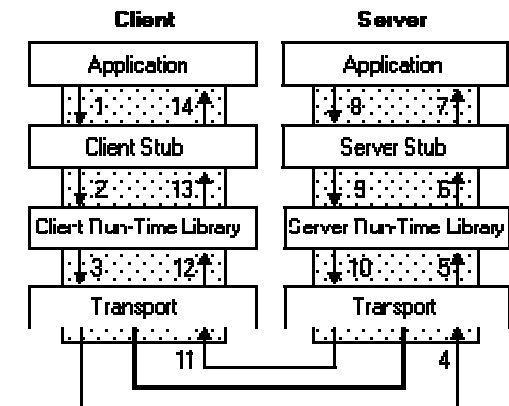
- q eine neue API wurde entworfen
 - q 330 Funktionen
 - q Implementation in PASCAL unter OpenVMS
 - q nicht netzwerkfähig
- q neue Clients werden mit C++ unter Windows NT/2000/XP entwickelt
- q Anbindung an das MEDAS-System über TCP/IP
- ∅ Die API muß netzwerkfähig gemacht werden und eine Win32-C(++)-Implementation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

Aufgaben einer Netzwerk-API

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

RPC

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



RPC: Implementierungen

- q zwei (inkompatible) Varianten:
 - q Distributed Computing Environment (DCE) RPC
 - q beim Win32 TCP/IP Stack dabei
 - q Runtime-Paket mit OpenVMS lizenziert
 - q Application Developers Kit kommerziell (QL-24GA*-AA)
 - q Open Network Computing (ONC) RPC
 - q SUNs RPC
 - q bei den OpenVMS TCP/IP Stacks dabei
 - q für Win32 sowohl kommerzielle als auch OpenSource Implementierungen

Ø Entscheidung zugunsten von ONC RPC

ONC RPC

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

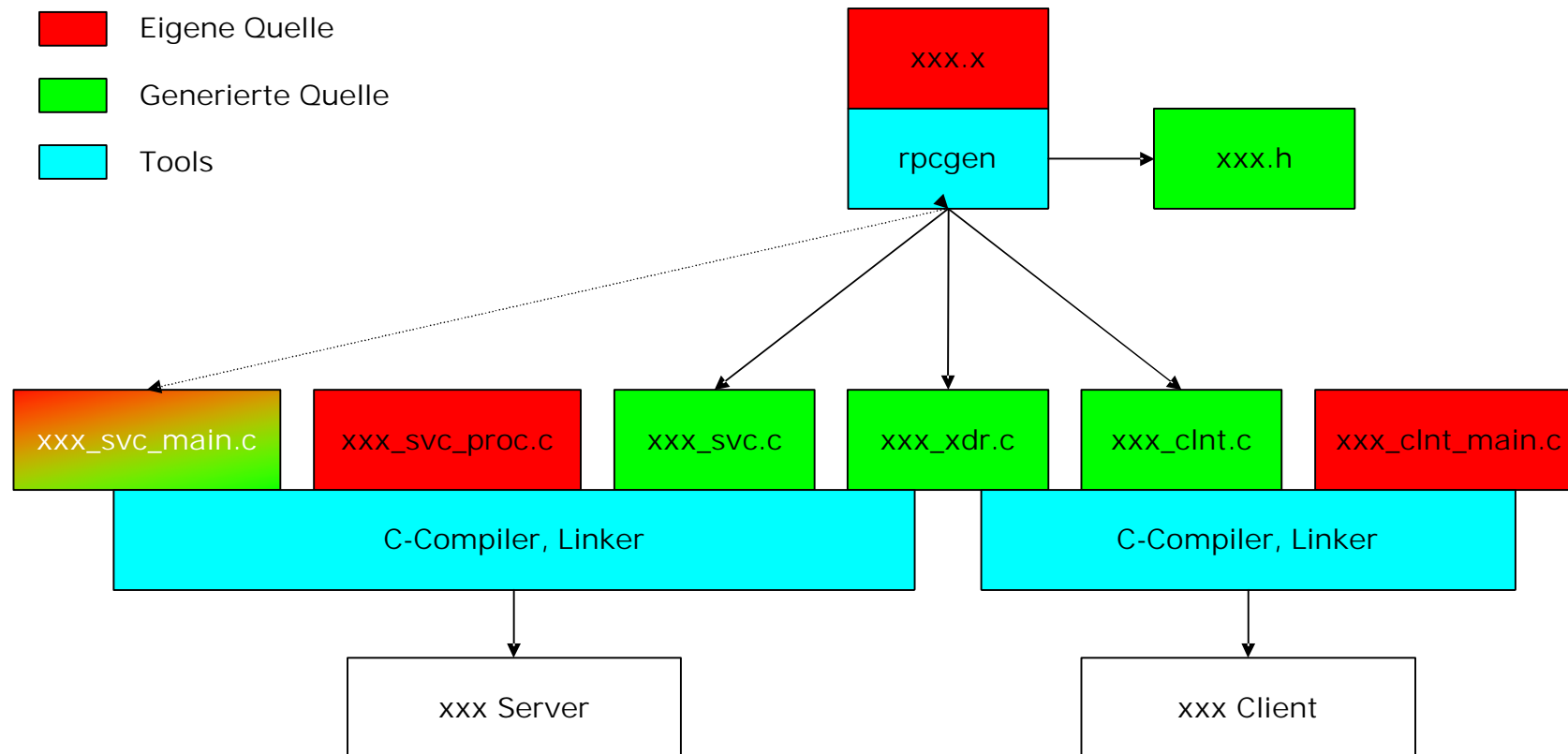
XDR

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

Boolean
[Unsigned] [Hyper] Integer
[Double-precision|Quadruple-precision] Floating-point
Enumeration
String

(Fixed-length|Variable-length) Array
Structure
Discriminated Union
(Fixed-length|Variable-length) Opaque Data
Void

RPC: Ein- und Ausgaben



An die Arbeit!

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

Beispiel einer Funktion

```
CONST
  Channel Li stLength = 3072;           {aus SRV: I NTERFACE_DATA. I NC}
TYPE
  TypeReturnCode = I NTEGER;           {aus SRV: I NTERFACE. I NC}
  TypeChannel Li st =                  {aus SRV: I NTERFACE_DATA. I NC}
    PACKED ARRAY [1..Channel Li stLength] OF CHAR;
  TypStatusByte = PACKED ARRAY [1..8] OF 0..1;      {aus GLB: TYPSTS. I NC}
  TypStatusRec = RECORD                {aus GLB: TYPSTS. I NC}
    Hardware_Status : TypStatusByte;
    MEDAS_Status    : TypStatusByte;
    Al arm_Status   : TypStatusByte;
    Aux_Status      : TypStatusByte;
  END;
  TypZei t = UNSI GNED;                {aus GLB: TYPTI M. I NC}
  TypeVal ueRecord = RECORD            {aus SRV: I NTERFACE_DATA. I NC}
    Val ue      : REAL;
    Status      : TypStatusRec;
    Ti meStamp  : TypZei t;
  END;

FUNCTION ReadMul ti pl eVal ues        {aus SRV: I NTERFACE_DATA. I NC}
( VAR Channel Li st : TypeChannel Li st;      {1}
  VAR Val ueCount   : I NTEGER;              {0}
  VAR Channel Array : ARRAY [j 0..j 1: I NTEGER] OF I NTEGER;      {0}
  VAR Val ueArray   : ARRAY [i 0..i 1: I NTEGER] OF TypeVal ueRecord {0}
) : TypeReturnCode; EXTERNAL;
```

m2srv.x (1/2)

```
const Channel Li stLength = 3072;

typedef i nt TypeReturnCode;

typedef u_char TypeChannel Li st[Channel Li stLength];

typedef u_char TypStatusByte[1];

struct TypStatusRec {
    TypStatusByte Hardware_Status;
    TypStatusByte MEDAS_Status;
    TypStatusByte Al arm_Status;
    TypStatusByte Aux_Status;
};

typedef u_i nt TypZei t;

struct TypeVal ueRecord {
    fl oat Val ue;
    TypStatusRec Status;
    TypZei t Ti meStamp;
};
```

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 ChannelLength 3072

typedef int TypeReturnCode;
bool_t xdr_TypeReturnCode();

typedef u_char TypeChannelList[ChannelLength];
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 ChannelArrayLen;
        int *ChannelArray_val;
    } ChannelArray;
    struct {
        u_int ValueArrayLen;
        TypeValueRecord *ValueArray_val;
    } ValueArray;
};
typedef struct res_ReadMultipleValues_t res_ReadMultipleValues_t;
bool_t xdr_res_ReadMultipleValues_t();
```

m2srv.h (3/3)

```
struct arg_ReadMultiplies_t {
    TypeChannelList ChannelList;
    struct {
        u_int ChannelArrayLen;
        int *ChannelArray_val;
    } ChannelArray;
    struct {
        u_int ValueArrayLen;
        TypeValueRecord *ValueArray_val;
    } ValueArray;
};
typedef struct arg_ReadMultiplies_t arg_ReadMultiplies_t;
bool_t xdr_arg_ReadMultiplies_t();

#define M2SRV_PROG 0x24242424
#define M2SRV_VERSION 1

#define ReadMultiplies ((u_long)141)
extern res_ReadMultiplies_t *readmultiplies_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_ReadMultiplies_t readmultiplies_1_arg;
        ...
    } argument;
    char *result;
    xdrproc_t _xdr_argument, _xdr_result;
    char *(*local)(char *, struct svc_req *);

    switch (rqstp->rq_proc) {
        ...
        case ReadMultiplies:
            _xdr_argument = (xdrproc_t) xdr_arg_ReadMultiplies_t;
            _xdr_result = (xdrproc_t) xdr_res_ReadMultiplies_t;
            local = (char *(*)(char *, struct svc_req *)) readmultiplies_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 *
readmultiple_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

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

Eine Grammatik-Regel

```
routi ne_decl arati on:
  /FUNCTION/i <commi t> i denti fi er parameter_l i st(?) ':' resul t_type_i d
  {
    $return =
    $::routi ne_decl {l c $i tem{i denti fi er}{__VALUE__}} =
      bless \%i tem, $i tem[0];
    $return->{__fi le__} = $::current_fi le;
  }
| /PROCEDURE/i <commi t> i denti fi er parameter_l i st(?)
  {
    $return =
    $::routi ne_decl {l c $i tem{i denti fi er}{__VALUE__}} =
      bless \%i tem, $i tem[0];
    $return->{__fi le__} = $::current_fi le;
  }
| <error?>
```

Die Grammatik

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

Die Struktur-Verarbeitung

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

Die Codegenerierung

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

```
package Rule : routine_declarati on;

use Rule : : identi fier;
use Rule : : parameter_li st;
use Rule : : resul t_type_id;

sub pascal_in c {
    my $sel f = shi ft;
    my $ret =
        $sel f->{__PATTERN1__} # /FUNCTI ON|PROCEDURE/
        ' '
        . $sel f->{i denti fier}->val ue;
    $ret .= $sel f->{parameter_li st}[0]->pascal_in c
        i f @{$sel f->{parameter_li st}};
    $ret .= ' : ' . $sel f->{resul t_type_id}->pascal_in c
        i f exists $sel f->{resul t_type_id};
    return $ret;
}
```


m2srv_svc_proc.c (1/2)

```
#undef ReadMultiPlValues

res_ReadMultiPlValues_t *
readmultiplvalues_1_svc(
    arg_ReadMultiPlValues_t * client_arg,
    struct svc_req * rqstp
) {
    static res_ReadMultiPlValues_t client_res;

    TypeChannelList ChannelList;
    CONFORMANT_ARRAY_STRUCT(jOj1, int, ChannelArray);
    CONFORMANT_ARRAY_STRUCT(iOi1, TypeValueRecord, ValueArray);
    TypeReturnCode _retCode;
    int ValueCount;

    /* Copy received data from RPC struct */
    memcpy(ChannelList, client_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 *)client_arg->ChannelArray.ChannelArray_val;
    ...
}
```

m2srv_svc_proc.c (2/2)

```
nca_Val ueArray. dsc. dsc$w_l ength = si zeof(TypeVal ueRecord);
nca_Val ueArray. dsc. dsc$b_dtype = DSC$K_DTYPE_Z;
nca_Val ueArray. dsc. dsc$b_cl ass = DSC$K_CLASS_NCA;
nca_Val ueArray. dsc. dsc$a_poi nter =
    (char *)cl nt_arg->Val ueArray. Val ueArray_val ;
...

_retCode = ReadMul ti pl eVal ues(
    &Channel Li st,
    &Val ueCount,
    &nca_Channel Array,
    &nca_Val ueArray
);

/* Copy data to send to RPC struct */
memcpy(&cl nt_res._retCode, &_retCode, si zeof(TypeReturnCode));
memcpy(&cl nt_res.Val ueCount, &Val ueCount, si zeof(i nt));
cl nt_res.Channel Array.Channel Array_l en =
    nca_Val ueArray. dsc$bounds[0]. dsc$I _u + 1;
cl nt_res.Channel Array.Channel Array_val =
    (i nt *)nca_Channel Array. dsc. dsc$a_poi nter;
cl nt_res.Val ueArray.Val ueArray_l en =
    nca_Val ueArray. dsc$bounds[0]. dsc$I _u + 1;
cl nt_res.Val ueArray.Val ueArray_val =
    (TypeVal ueRecord *)nca_Val ueArray. dsc. dsc$a_poi nter;
return &cl nt_res;
```

```
}
```

m2srv_clnt_proc.c (1/2)

```
CLIENT * client;

#undef ReadMultiValues

TypeReturnCode ReadMultiValues(
    /*R*/ TypeChannelList * ChannelList,
    /*W*/ int * ValueCount,
    /*M*/ CONFORMANT_ARRAY_PARAMETER(j Oj 1, int, ChannelArray),
    /*M*/ CONFORMANT_ARRAY_PARAMETER(i Oi 1, TypeValueRecord, ValueArray)
) {
    arg_ReadMultiValues_t client_arg;
    res_ReadMultiValues_t * client_res;
    TypeReturnCode _retCode;

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

    /* Copy data to send to RPC struct */
    memcpy(client_arg.ChannelList, *ChannelList, sizeof(TypeChannelList));
    client_arg.ChannelArray.ChannelArray_1en = ChannelArray_1en;
    client_arg.ChannelArray.ChannelArray_val = ChannelArray_val;
    client_arg.ValueArray.ValueArray_1en = ValueArray_1en;
    client_arg.ValueArray.ValueArray_val = ValueArray_val;
}
```

m2srv_clnt_proc.c (2/2)

```
clnt_res = readmultiplervalues_1((void*)&clnt_arg, client);
if (clnt_res == (res_ReadMultiplervalues_t *) NULL) {
    clnt_perror(client, "ReadMultiplervalues");
    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)

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

Stolpersteine (2/3)

- q SET OF enum-type
 - q Umgesetzt als char[] plus enum, der Masken enthält
- q Conformant Array-Parameter:
ARRAY [i0..i1 : INTEGER] OF type
 - q RPC-Typ "variable length array"
 - q wegen Problemen mit der TCPware-RPC-Implementation verworfen
 - q statt dessen: Änderung der API auf feste Arrays
- q Maximale Länge von Identifiern beim VMS-Linker: 31 Zeichen
 - q Verkürzung der Identifier auf 21 Zeichen, z.B. GetPCSCConnectionCharacteristics nach GetPCSCConnectionCha_1, so daß xdr_arg_getpcsconnectioncha_1_t im Limit bleibt

Stolpersteine (3/3)

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

q Eingabe:

- q 97 aus 320 PASCAL-Include-Dateien
- q 974 Konstanten
- q 542 Typen
- q 744 Routinen

q Ausgabe:

- q 252 Konstanten
- q 245 Typen
- q 330 Routinen

Vorher...

