EFDA Task Force

Integrated Tokamak Modelling

EUROPEAN FUSION DEVELOPMENT AGREEMENT

Using XML for Code Specific Parameters

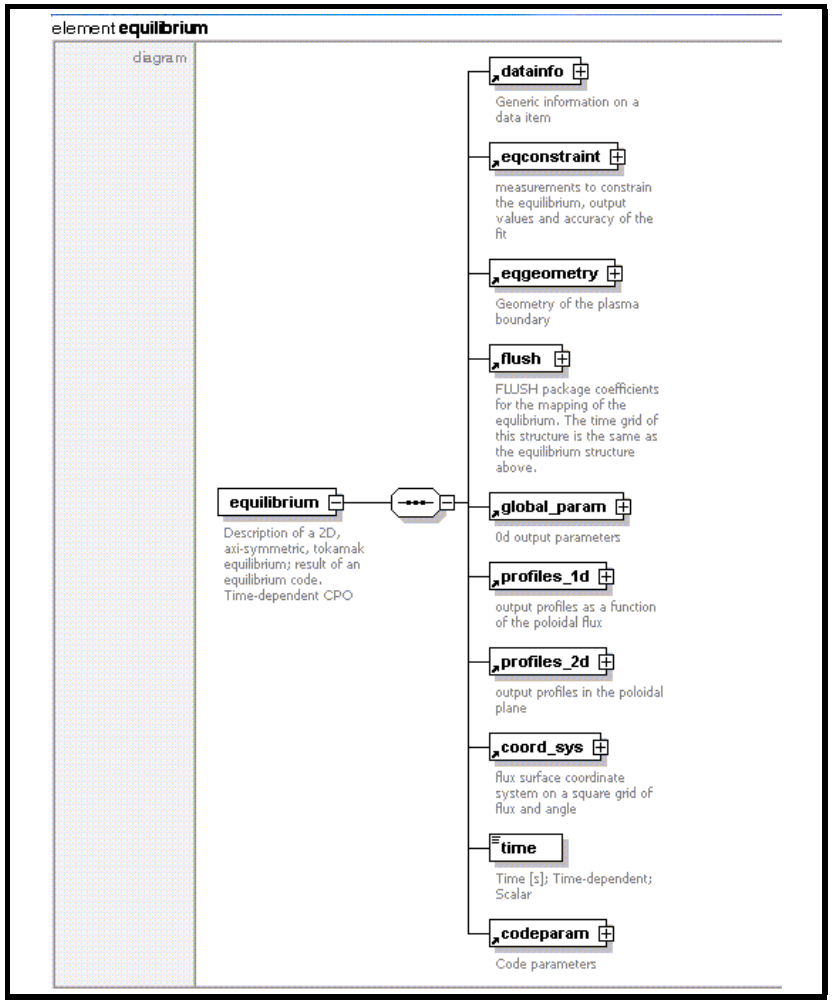
C. Konz

ITM TF Kepler Training, Cadarache
May 4, 2009

Outline:

- Code Specific Parameters
- Proposed Approach
- W3C XML Schemas and F95 XML Parser
- Tools for Autogeneration of Schemas

XML Use for ITM Data Structures



Motivation:

- structures form the basis of objects (Consistent Physical Objects (CPOs))
- organize data exchange between codes and database
- tree-like hierachical structures
- need for a highly flexible data format → XML
- need for self-descriptive format → XML Schemas
- need for a convenient way of generating Fortran and C include files → XSLT

Code Specific Parameters

All parameters which are specific to the code (like switches, scaling parameters, and parameters for built-in analytical models) as well as parameters to explicitly overrule fields in the ITM data structures. Generally no data (should go into CPOs).

Proposal:

As the rest of the data structure, all code specific parameters should be given in XML format, i.e., in form of an XML string.

```
type type_codeparam  !    Code parameters
  character(len=132), dimension(:), pointer ::codename  ! /codeparam/codename - Name of the code
  character(len=132), dimension(:), pointer ::codeversion  ! /codeparam/codeversion - Version of the code (as in the ITM repository)
  character(len=132), dimension(:), pointer ::parameters  ! /codeparam/parameters - List of the code specific parameters, string expected
  character(len=132), dimension(:), pointer ::output_diag  ! /codeparam/output_diag - List of the code specific diagnostic/output, string
endtype
```

Example: input_ilsa.xml

```
<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="./input_ilsa.xsl"
charset="ISO-8859-1"?>

<parameters>

<!-- mode -->

  <mode>
    <version> eigenvalue problem </version>
    <modus> mishka_1 </modus>
    <solver> inverse vector iteration </solver>
  </mode>

<!-- ... and so on ... -->

<!-- boundary conditions -->

  <boundary_conditions>
    <boundary> free boundary </boundary>
    <rwall> 2.50 </rwall>
    <shape_ideal> plasma boundary </shape_ideal>
    <mvanz> 51 </mvanz>
    <nvpsi> 101 </nvpsi>
    <imeshac_vac> yes </imeshac_vac>
    <acc_type_vac> 0 </acc_type_vac>
    <equidistant_vac> 0 </equidistant_vac>
  <!-- 1:plasma boundary, 2:antenna, 3:resistive wall, 4:ideal wall -->
    <mesh_accum_vac> yes       no          no              yes </mesh_accum_vac>
    <ds_min_vac>     0.001     0.0001      0.001           0.001 </ds_min_vac>
    <c_s_vac>        0.01      0.1         0.1             0.01 </c_s_vac>
  </boundary_conditions>

</parameters>
```

Why XML?

- extremely versatile markup language ('generalisation' of HTML)
- self-describing data through use of DTDs (document type definitions) or schemas
- simple to edit: plain ASCII, similar to HTML
- but can handle all levels of complexity
- large and fast growing user community
- large infrastructure of tools for XML creation, manipulation, and usage: XPath, XPointer, XSLT, XSL-FO, CSS, parsers, editors, browsers, etc.
- already in use for CPO definitions
- allows separation of generic tools and code specific parameters

Proposed Approach:

Step 1: Strip structure of code specific parameters (i.e. names, types, structures, dimensions, allowed choices and ranges, etc.) into a separate file, a so-called **W3C XML Schema**.

Advantages:

- no format specific read subroutines needed anymore
- all tools can be made generic
- all code specific information stored in one single external file
- creation of the schema is a 'once-in-a-code's-lifetime' event
- later changes very simple through changing the schema
- enables input checking before running the code
- schema serves as minimum documentation for input

Proposed Approach:

Step 2: Convert former input files into **XML files** which are instances of the XML schema of the code.

Advantages:

- text input files easier to understand by user
- same advantage as namelist: input does not have to be complete
- free order of input parameters as long as structure is not changed
- possibility to define beginner's and expert's settings
- input checks possible
- XML can be used for namelist input as well as any other format

Proposed Approach:

Step 3: Use generic tools **(F95 XML Parser, string manipulation toolbox, CREATE_SCHEMA, CREATE_ASSIGN, schema based GUIs, open source XML tools)** to automatically generate W3C schemas, modify code specific parameters and read them in from a file/database.

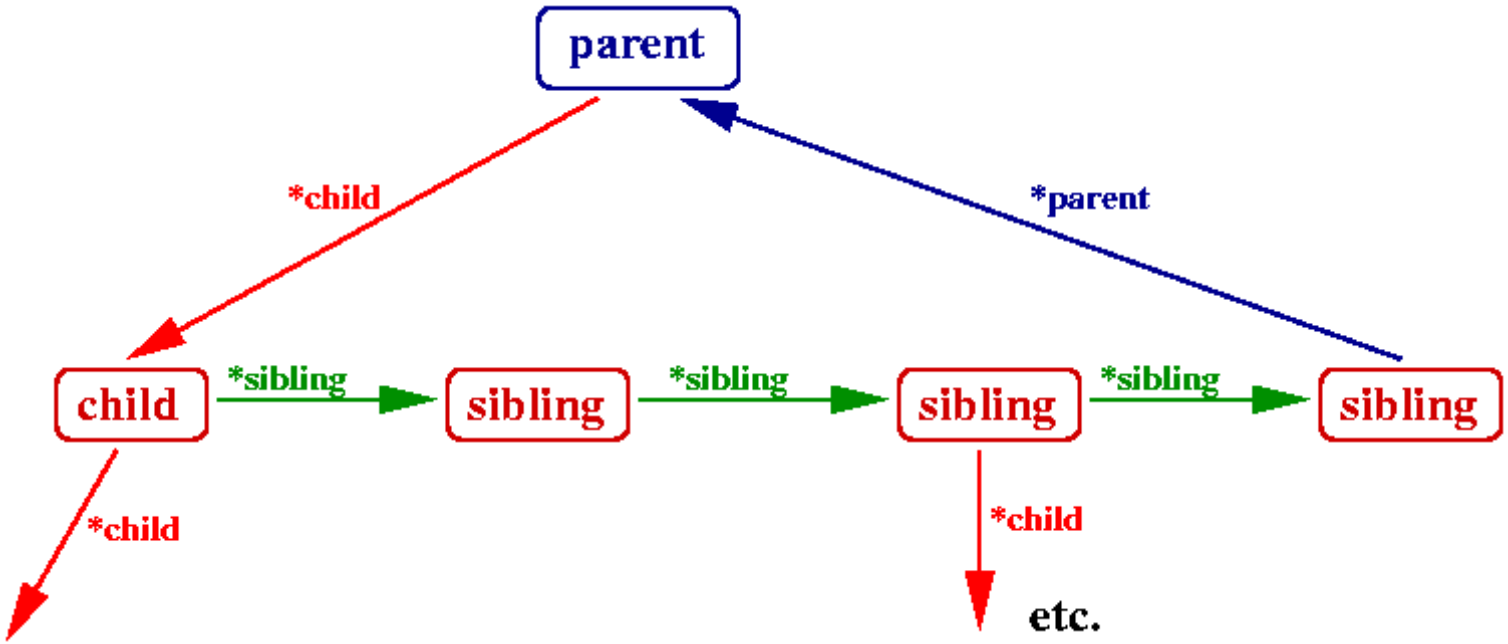
Advantages:

- generic tools have to be developed only once and can be used for any code
- generic tools as separate library - easier to maintain
- GUI development or use of existing GUIs become possible (e.g. xforms in a browser as suggested by G. Huysmans)
- users do not need to know about XML at all
- developers need to know only very little about XML
- tools for creation of a skeleton XML schema and the required assignment subroutine in Fortran are available

Lightweight FORTRAN parser for XML documents:

- compact (< 600 FORTRAN lines), efficient, and fast parser
- parses XML documents with arbitrary depth and complexity (except for attributes)
- based on W3C XML Schemas (can be used to validate XML documents)
- uses tree-like lists with parent, child, and sibling pointers
- possible to incorporate namelists (FORTRAN read from strings)
- tag names and value lists of arbitrary length (dynamical memory allocation)
- available as module `euitm_xml_parser` (no dependencies)
- first parses the code specific W3C schema, then parses the entire XML document sequentially like SAX
- comes with useful subroutines in `xml_tools.f90` and `string_manipulation_tools.f90` (to be expanded)

Tree Structure



euitm_xml_parse:

- parses the schema and builds an empty tree with the structure described by the schema: associates the corresponding pointers, allocates the tag names `cname` and fills in the tag names
- parses the actual XML document and fills the parsed values `cvalue` into the tree
- returns the complete tree in `parameter_list` and the number of successfully parsed parameters `nparm`

Calling the Fortran XML Parser:

```
  use euitm_schemas
  use euitm_xml_parser

  implicit none

  type (type_codeparam), intent(in) :: codeparameters
  integer(ikind), intent(out) :: return_status

  type(tree) :: parameter_list
  type(element), pointer :: temp_pointer
  integer(ikind) :: i, nparm, n_values
  character(len = 132) :: cname

!-- set path to XML schema
  file_xml_schema = 'helena_schema.xml'

  return_status = 0       ! no error

!-- parse xml-string codeparameters%parameters

  call euitm_xml_parse(codeparameters%parameters, nparm, parameter_list)

!-- assign variables

  temp_pointer => parameter_list%first

  outer: do
    cname = char2str(temp_pointer%cname)    ! necessary for AIX
    select case (cname)
      case ("parameters")
        temp_pointer => temp_pointer%child
        cycle
```

assign_code_parameters:

- sets pointer to head of list `parameter_list%first`
- assigns values to in-code variables by stepping through the tree and using `select case` constructs and the interfaces in `string_manipulation_tools.f90`
- finally destroys the tree

```
!-- assign code parameters to internal variables
  call assign_code_parameters(euitm_equilibrium_out(1)%codeparam, &
   return_status)

  if (return_status /= 0) then
    write(iu6, *) 'ERROR: Could not assign code parameters.'
    return
  end if
```

Creating Schemas with CREATE_SCHEMA:

```
-+shot                                  ! shot related parameters
  source          string*15             ! source specifier
  nshot           integer
  tshot           float                 ! time slice
-+options                               ! option parameters
  udsym           boolean               ! up-down symmetric
  psibnd          float
  mfm             integer
```

Create a parameter list in `parameter_list.txt`:

- precede namelist names with '-+' level identifiers
- list name, type, and dimension for each parameter in namelist
- specify length of stringths with '*' right after 'string' type
- add comments following '! '
- names must be alphanumeric, no special characters, no spaces, underscores allowed

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- document element -->
  <xs:element name="parameters">
    <xs:complexType>
      <xs:all>
        <xs:element ref="shot"/>
        <xs:element ref="options"/>
      </xs:all>
    </xs:complexType>
  </xs:element>

  <!-- shot related parameters -->
  <xs:element name="shot">
    <xs:complexType>
      <xs:all>
        <xs:element ref="source"/>
        <xs:element ref="nshot"/>
        <xs:element ref="tshot"/>
      </xs:all>
    </xs:complexType>
  </xs:element>

  <!-- option parameters -->
  <xs:element name="options">
    <xs:complexType>
      <xs:all>
        <xs:element ref="udsym"/>
        <xs:element ref="psibnd"/>
        <xs:element ref="mfm"/>
      </xs:all>
    </xs:complexType>
  </xs:element>

  <!-- source specifier -->
  <xs:element name="source">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:maxLength value="15"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
```

Run CREATE_SCHEMA:

- move `parameter_list.txt` into `input/`
- `gmake -f makefile_pgi` in `obj/`
- `./create_schema.e` in `run/`
- $\Rightarrow$ `w3c_schema.xsd` in `output/`

Polish and improve your schema (optional):

- add `minOccurs="0"` if parameter is optional
- add range restrictions for integers and floats by defining new simpleTypes using restriction with minInclusive and maxInclusive or minExclusive and maxExclusive
- define allowed options for strings or integers using pattern
- limit length of arrays by using maxLength
- etc., etc.

```
subroutine assign_code_parameters(codeparameters, return_status)

!------------------------------------------------------------------------
! calls the XML parser for the code parameters and assign the
! resulting values to the corresponding variables
!------------------------------------------------------------------------

  use itm_types

!Add the modules hosting the relevant variables here!

  use euitm_schemas
  use euitm_xml_parser

  implicit none

  type (type_codeparam), intent(in) :: codeparameters
  integer(ikind), intent(out) :: return_status

  type(tree) :: parameter_list
  type(element), pointer :: temp_pointer
  integer(ikind) :: i, nparm, n_values
  character(len = 132) :: cname

!-- set path to XML schema
  file_xml_schema = 'my_schema.xsd'

  return_status = 0      ! no error

!-- parse xml-string codeparameters%parameters

  call euitm_xml_parse(codeparameters%parameters, nparm, parameter_list)

!-- assign variables

  temp_pointer => parameter_list%first

  outer: do
    cname = char2str(temp_pointer%cname)    ! necessary for AIX
    select case (cname)
      case ("parameters")
        temp_pointer => temp_pointer%child
        cycle
      case ("shot")
        temp_pointer => temp_pointer%child
        cycle
      case ("source")
        if (allocated(temp_pointer%cvalue)) &
         source = char2str(temp_pointer%cvalue)
```

Run CREATE_ASSIGN:

- move `parameter_list.txt` into `input/`
- move `w3c_schema.xsd` into `input/`
- `gmake -f makefile_pgi` in `obj/`
- `./create_assign.e` in `run/`
- $\Rightarrow$ `assign_code_parameters.f90` in `output/`

Resources:

Join project XMLLIB under GForge:

http://gforge.efda-itm.eu/

Fortran 90 XML Parser:

http://gforge.efda-itm.eu/svn/xmllib/trunk/parser

Schema Generator CREATE_SCHEMA:

http://gforge.efda-itm.eu/svn/xmllib/branches/create_schema

Generate assign_code_parameters.f90 with CREATE_ASSIGN:

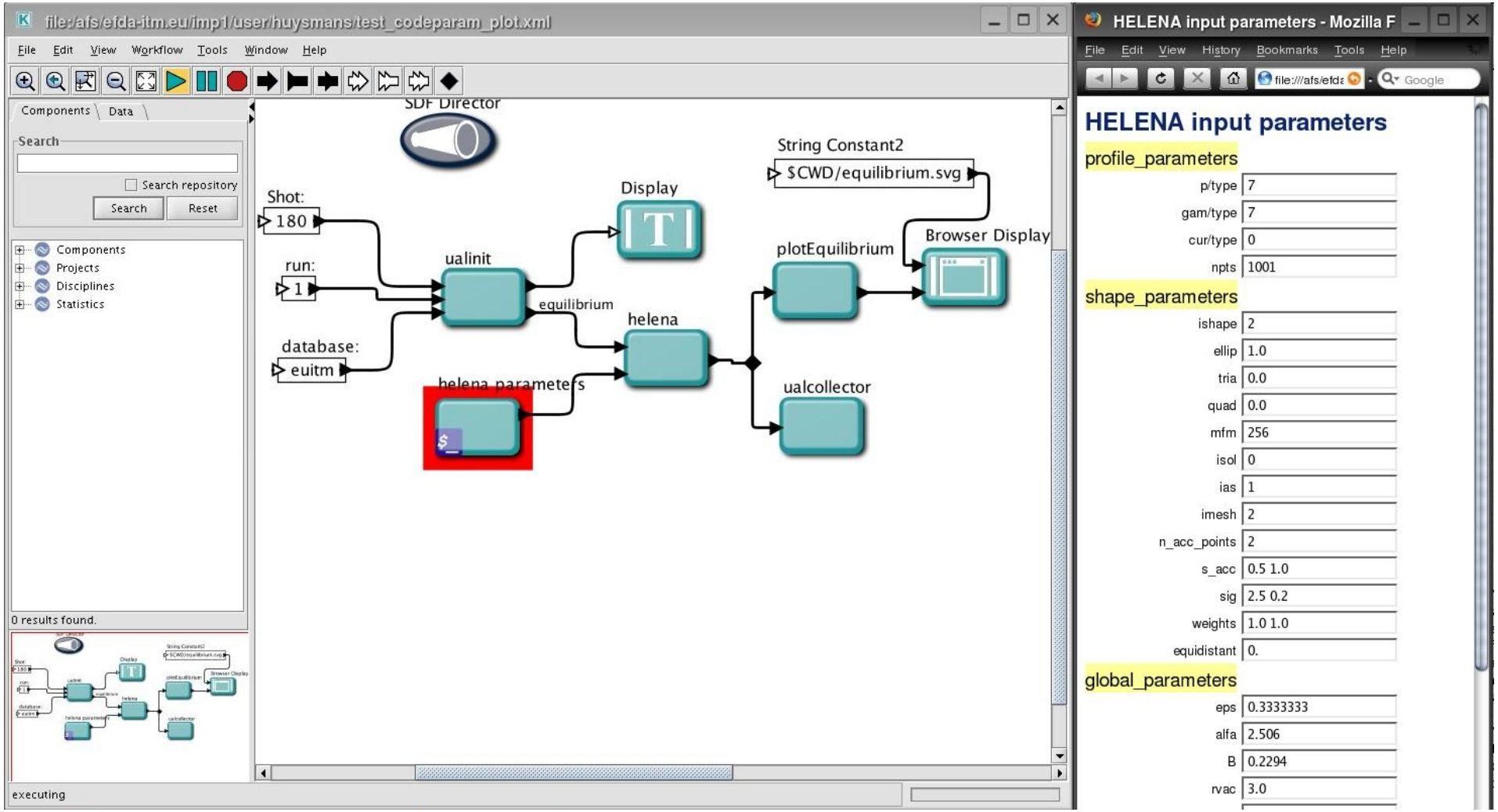
http://gforge.efda-itm.eu/svn/xmllib/branches/create_assign

GUI for Code Specific Parameters (proposal by G. Huysmans):

Use browser to interface XML documents based on W3C schema.

XForms: 'XML application that represents the next generation of forms for the Web. By splitting traditional XHTML forms into three parts - XForms model, instance data, and user interface - it separates presentation from content...' (W3C)

- takes a W3C XML schema to generate the fields in the form
- uses an XML document to fill data into the fields (default values possible)
- XForm is an XHTML file created from the XML schema using a stylesheet (done only once)
- existing extensions to Firefox 2 and 3 required



(by courtesy of G. Huysmans)

XML Tools:

tons of open source XML tools available

xmlstarlet carries out various XML operations, including validation against DTDs and schemas (`http://xmlstar.sourceforge.net/`).

Examples:

To test whether a file is well-formed XML:

```
xml val -w input_helena.xml
```

To test a file against an XML schema:

```
xml val -e --xsd helena_schema.xml input_helena.xml
```

IBM XML Schema Quality Checker:

Checks for problems in W3C XML Schemas, and clearly identifies any problems found (`http://www.alphaworks.ibm.com/tech/xmlsqc`).

To check a schema file:

```
ibmsqc helena_schema.xml
```

Short Introduction to W3C XML Schemas:

- Among other schemas (RELAX NG, Schematron) **W3C XML Schemas** are directed toward describing how elements are arranged in a document (like the syntax or grammar of a language - your personal 'XML language').
- Other than DTDs, W3C XML Schemas can also constrain the type of data in an element.
- XML Schemas are themselves XML documents, i.e., allow for checks for well-formedness and validity

Example: root element and container elements

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- document element -->
  <xs:element name="parameters">
    <xs:complexType>
      <xs:all>
        <xs:element ref="mode" maxOccurs="1"/>
        <xs:element ref="numerical_parameters" maxOccurs="1"/>
        <xs:element ref="plot_data" maxOccurs="1"/>
        <xs:element ref="physical_parameters" maxOccurs="1"/>
        <xs:element ref="boundary_conditions" maxOccurs="1"/>
        <xs:element ref="external_perturbation" minOccurs="0" maxOccurs="1"/>
      </xs:all>
    </xs:complexType>
  </xs:element>

  <!-- mode -->
  <xs:element name="mode">
    <xs:complexType>
      <xs:all>
         <xs:element ref="version" maxOccurs="1"/>
         <xs:element ref="modus" maxOccurs="1"/>
         <xs:element ref="solver" maxOccurs="1"/>
         <xs:element ref="toroidal_mode_number_scan" minOccurs="0"
                     maxOccurs="1"/>
         <xs:element ref="toroidal_scan_mode" minOccurs="0" maxOccurs="1"/>
         <xs:element ref="iterative_scan" minOccurs="0" maxOccurs="1"/>
         <xs:element ref="iterative_scan_mode" minOccurs="0" maxOccurs="1"/>
         <xs:element ref="mode_type" minOccurs="0" maxOccurs="1"/>
         <xs:element ref="poloidal_window" minOccurs="0" maxOccurs="1"/>
         <xs:element ref="equilibrium" maxOccurs="1"/>
         <xs:element ref="in_equilibrium" maxOccurs="1"/>
         <xs:element ref="format_type" maxOccurs="1"/>
         <xs:element ref="in_units" maxOccurs="1"/>
         <xs:element ref="out_length" maxOccurs="1"/>
         <xs:element ref="out_num" maxOccurs="1"/>
         <xs:element ref="stop_program" maxOccurs="1"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
```

Example: string with prescribed values

```
<xs:element name="version">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="eigenvalue problem|external perturbation"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example: string with length constraint

```
<xs:element name="in_equilibrium">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example: integer with minimum value

```
<xs:element name="manz">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example: user defined simpleType unit_float

```
<xs:simpleType name="unit_float">
  <xs:restriction base="xs:float">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="1"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="r_beg" type="unit_float"/>
```

Example: array of user defined simpleType

```
<xs:element name="s_min" type="RestrictedUnitFloatList"/>
<xs:element name="ds_min" type="RestrictedUnitFloatList"/>
<xs:element name="c_s" type="RestrictedPosFloatList"/>

<xs:simpleType name="PosFloatList">
   <xs:list itemType="pos_float"/>
</xs:simpleType>

<xs:simpleType name="RestrictedPosFloatList">
  <xs:restriction base="PosFloatList">
    <xs:maxLength value="10"/>
  </xs:restriction>
</xs:simpleType>
```

Limitation: no arrays of complex types allowed!

(issue with arrays of complex numbers for instance; may hopefully be lifted in the near future)