4 Standardization

Stefan Benke and Georg J. Schmitz

4.1 Overview

Standardization in the context of the AixViPMaP platform currently covers two different aspects. In the first step, the simulation of manufacturing process chains requires the exchange of state variables describing the history of the material or component along the entire value chain from casting of the homogeneous melt to the application of the final product. The simulation of the individual process steps usually is performed by specialized simulation tools, which might be academic ''in-house,'' commercial software packages, proprietary industrial developments, or open source codes. In order to enable a seamless integration of the tools into the platform and the use of one common postprocessing tool for all simulation software packages, a standardization of the result data, geometry data, material data, and boundary conditions is required.

In the second step, which intervenes deeper into the different individual software tools, the standardization efforts extend to functional dependencies for modeling of the specific material behavior. In order to exchange the functional description of the material models between the simulation codes and to enable the realization of a strongly coupled, scale bridging simulation using different software codes on all involved length scales, an application programming interface (API) for user-defined material subroutines has to be standardized. These user subroutines may be used to model the thermal, mechanical, electrical, and metallurgical behavior of the material.

Common to all standardization activities on the AixViPMaP platform is the usage of very simple file formats and programming methods in order to keep the threshold for participation as low as possible. As most of the existing simulation tools in computational engineering of materials are – because of historical reasons – programmed in programming languages such as Fortran or C, the exchange of simulation data is easily realized by the use of plain ASCII or binary files rather than using databases or object-oriented programming methods. The additional advantage of this approach is that all basic file operations are supported by most of the various operation systems currently used.

Integrative Computational Materials Engineering: Concepts and Applications of a Modular Simulation Platform,
First Edition. Edited by Georg J. Schmitz and Ulrich Prahl.

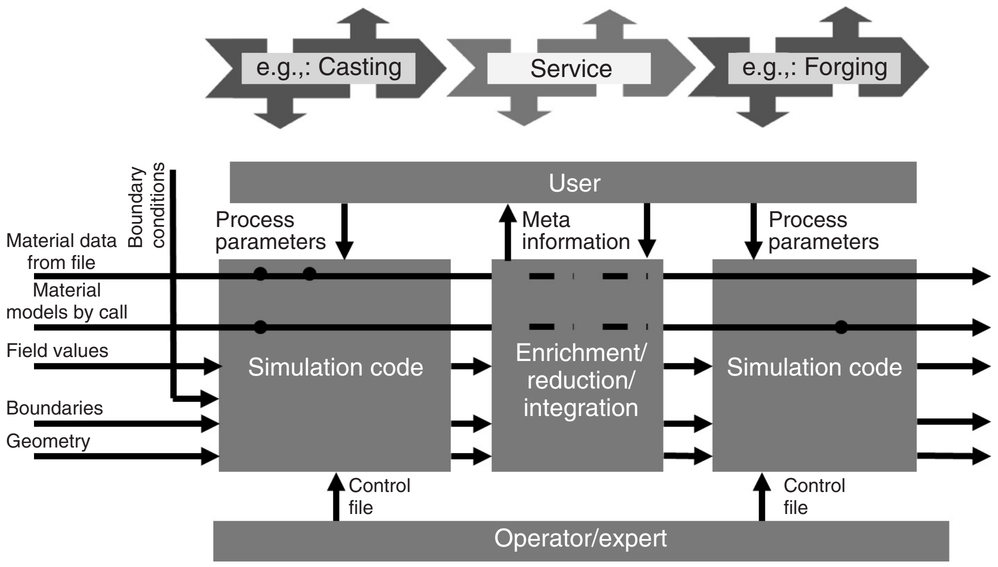

Figure 4.1 Scheme of the bus-type information flow between two different modules of the AixViPMaP® simulation platform. Individual simulation tools are daisy chained along the different processes steps of the entire production process. The information exchange is file based, using VTK files for geometry, boundaries, and field values, ASCII files for material data from file.

This chapter documents preliminary standards for the file-based data exchange and the API. In view of the ongoing exploration of the possibilities of the platform, the current state of the documentation serves as a very basic standard. An updated version may be found at the AixViPMaP site [1]. In the long term, further development should become a cooperative effort of an international community, for example, in the frame of the TMS committee on integrated computational materials engineering (ICME).

An overview of the standardization efforts to be taken is depicted in Figure 4.1. The following sections describe the present status with respect to standardization of geometry and APIs.

4.2 Standardization of Geometry and Result Data

The description of the discretized component geometry on the macroscale and the morphology of the microstructure on the microscale are both based on the file format of the Visualization Toolkit (VTK) [2]. VTK is an open source toolkit for the visualization of discretized data stemming from various sources such as numerical simulations, computed -tomography, or others. The toolkit is distributed by Kitware Inc. and is basically written in C++. It provides a number of predefined data structures and methods for the handling of data files, the manipulation of mesh and result data, and their visualization. A number of software applications based on VTK are available as open source software, highlighting ParaView [3] as one of the most popular tools for data visualization, especially in 3D. This leads to the fact that the VTK-file format – besides netCDF and HDF – is one of the most popular open file formats for numerical result data.

Since the VTK-file format is a very basic and universal file format for discretized data, several extensions and standardizations are needed while keeping the compatibility to the basic format. The missing data in the file are especially the information on previous simulation steps, unit systems, coordinate systems, and timing data. These have now been included into an extended file header, as described in the next section. The standardization issues are mostly related to the nomenclature of the datasets containing group and result data. The rest of the file coincides with the ASCII or binary version of "legacy file format," which is documented in detail elsewhere [4] and is briefly recalled here. In general, the file consists of five basic parts [4]:

1) The first part contains the file version and the identifier of the file format.
2) The second part is the header. It consists of a character string that may be 256 characters long. On the AixVipMap platform, this character string is used to transport information about process step and process time as well as about the unit system (Section 4.2.1).
3) The third part defines either the ASCII or binary file type format. Both types are used on the AixViPMaP platform.
4) The fourth part comprises information about the geometry and topology of the dataset. Currently, only discretized geometries are used on the platform. The discretization types currently being used are structured points, mainly for microstructure descriptions from phase-field simulations; structured grids; and unstructured grids, mainly for macroscopic finite element (FE) process simulations.
5) The last part of the file contains the dataset attributed or field variables, which may be scalar-, vector-, or tensor-type data. The keywords describing the attribute values are standardized in Section 4.2.3. The field variables are also used to define geometric group information, as defined in Section 4.2.2.

The naming of the result files follows a specific naming scheme in order to organize the result files in a meaningful manner. The name of the result files consists of the English name of the individual process step followed by a sequential number counting the number of simulations using this type of process step, a dot, followed by the sequential number of the time step writing this process step, and eventually the extension ".vtk." For example, the name of the result file stemming from the second heating step within a process chain is given by: `heating02.10.vtk`. Here, the number 10 refers to the 10th time step of the process step.

4.2.1 Extended File Header

The header of the generic VTK-file is extended on the AixViPMaP platform in order to store the information on the simulation step, which may be used on a higher abstraction level and for postprocessing purposes. The header section at the beginning of the VTK-file looks as follows:

```
#vtk DataFile Version 2.0
process_step time=n genericSI|modifiedSI
```

Where the process_step is the English name of the simulation process step, time is the current simulation time in time units as given by the unit system. The *unit system* is defined either as generic SI unit system or a modified SI system using arbitrary units. For a modified SI unit system, the conversion factors to the SI unit system must be defined as follows:

```
process_step time=n modifiedSI l=1.0E-06 f=1.0
t=1.0 dt=273.15 p=1.0 to=0.0 c=wt
```

For non-SI unit systems, the factors for converting the field variables and measurements into SI units must be declared. They are defined as follows:

`l`	Scaling factor from length units to SI units (meter)
`f`	Scaling factor for forces to SI units
`t`	Scaling factor for temperature units
`p`	Scaling factor for time periods
`dt`	Temperature offset to 0 K in temperature units
`to`	Time offset in time units
`c =wt|at`	Defines either mass or atom percent for species concentration

If a factor is not given, it is implicitly assumed to be a unit factor or a zero offset.

4.2.2 Geometric Attributes

Geometric attributes are currently used to store information about the groups of geometric entities. Thus, they are used to define a subset of the discretized cells or points of the geometry under consideration. In FE analysis, the definition of point or cell groups is very common. They are mostly used to define geometric regions for the application of initial and boundary conditions. As the VTK-file format does not provide a special data structure for the storage of groups, they are defined as attribute field datasets on points or cells.

Material ID The attribute field `Material ID` is intended to decompose a part or context geometry into several geometric regions that consist of different materials. The attribute field is stored as a generic "`CELL DATA`" field as an integer number. The number corresponds with the identifier of the material. All material identifiers should – but need not – be sequentially numbered.

Groups of Points The point groups are stored as an attribute dataset on points. The naming convention for these groups is defined such that the name should start with the prefix "`POINT_GROUP_ ...` ". Here, the dots are placeholders for the unique naming suffix. The *attribute values* are defined as integer values for all points of the model. If a point belongs to the group, its attribute value should not be zero. All other points not being members of the actual group should have a zero attribute value. It is advised that all point groups should be sequentially numbered and that this number should be used as the attribute value in order to allow set operations on the point groups.

Groups of Cells The cells are defined in the same way as the groups of points. The naming scheme for cell groups starts with the prefix "`CELL_GROUP`" followed by a unique naming suffix. Both parts of the name must be connected by an underscore. If a cell is a member of a group, its integer attribute value should not be zero. All other cells must have a zero attribute value. Here again, it is advised that all cell groups should be sequentially numbered and that this number should be used as the attribute value in order to allow set operations on the point groups.

4.2.3 Field Data

In the following, the names of the field datasets are defined and standardized in order to enable an automatic processing of the data by the different simulation tools and the enricher/parser software. Basically, the name of the field data is given by the English name of the field variable. The names are case insensitive but may not be separated by whitespace characters. Data items not being updated in the current simulation step, for example, because of a lack of material model, should be passed unchanged to the following simulation step (Table 4.1).

4.3 Material Data

Different types of material data – especially thermodynamic and thermophysical – are relevant to ICME. Thermodynamic and kinetic data are relevant, for example, to model microstructure evolution and also to account, for example,

Table 4.1 Standardized names for the field variables in attribute datasets.

Entity	Type	SI unit
backstress	Tensor	Pa
concentration_*Xx*[a]	Scalar	wt%|at%
dislocation_density	Scalar	$1\,m^{-2}$
displacement	Vector	M
enthalphy[b]	Scalar	$J\,m^{-3}$
equivalent_strain_rate	Scalar	$1\,s^{-1}$
equivalent_strain	Scalar	1
equivalent_plastic_strain	Scalar	1
equivalent_stress	Scalar	Pa
fraction_crystalline[c]	Scalar	1
heat_flux	Vector	$J\,mm^{-2}$
hydrostatic_strain	Scalar	1
hydrostatic_stress	Scalar	Pa
grain_size[d]	Scalar	M
grain_size_min	Scalar	M
grain_size_max	Scalar	M
nodal_force	Vector	N
material_orientation[c]	Tensor	1
phase_frac_*name*[e]	Scalar	1
plastic_strain	Tensor	1
spherulite_number[c]	Scalar	1
spherulite_diameter[c]	Scalar	M
stiffness_hooke	Tensor	Pa
strain	Tensor	1
strain_energy[f]	Scalar	J
strain_energy_density	Scalar	$J\,m^{-3}$
strain_rate	Tensor	$1\,s^{-1}$
temperature	Scalar	K
temperature_rate	Scalar	$K\,s^{-1}$
velocity	Vector	$m\,s^{-1}$
shear_rate[c]	Vector	$1\,s^{-1}$
liquidus_temperature	Scalar	K
solidus_temperature	Scalar	K
vaporisation_temperature	Scalar	K
heat_of_fusion	Scalar	$J\,m^{-3}$
heat_of_condensation	Scalar	J/m^3
density_at_rt	Scalar	$kg\,m^{-3}$
density_at_liquidus	Scalar	$kg\,m^{-3}$
density_at_solidus	Scalar	$kg\,m^{-3}$
density_at_vapor	Scalar	$kg\,m^{-3}$
rho_cp_at_rt	Scalar	$J\,m^{-3}\,K$
rho_cp_at_liquidus	Scalar	$J\,m^{-3}\,K$
rho_cp_at_solidus	Scalar	$J\,m^{-3}\,K$
rho_cp_at_vapor	Scalar	$J\,m^{-3}\,K$
specific_heat_mass_at_rt	Scalar	$J\,kg^{-1}\,K$

Table 4.1 (*continued*)

Entity	Type	SI unit
specific_heat_mass_at_liquidus	Scalar	J/kg K
specific_heat_mass_at_solidus	Scalar	J kg^{-1} K
specific_heat_mass_at_vapor	Scalar	J kg^{-1} K
thermal_conductivity_at_rt	Scalar	W m^{-2} K
thermal_conductivity_at_liquidus	Scalar	W m^{-2} K
thermal_conductivity_at_solidus	Scalar	W m^{-2} K
thermal_conductivity_at_vapor	Scalar	W m^{-2} K

[a] *Xx* is the abbreviation of the name of the chemical species.
[b] Stored enthalpy of the material at a given temperature.
[c] Only for plastics/polymers.
[d] Average grain size for a macroscale process simulation.
[e] Where `name` is the English denomination of the phase, for example, austenite, ferrite, bainite, srx (static recrystallized), drx (dynamically recrystallized), mrx (metadynamically recrystallized), and so on.
[f] Total strain.

for phase fractions, enthalpies, or other thermodynamic values being required as effective local values for simulations at the process scale. The standardization of such data has already largely taken place within the CALPHAD community in the form of the widespread thermodynamic database "*.tdb" file format. Many databases for technical alloy systems are commercially available and can be accessed by APIs provided by the respective software companies. Present activities aim at generating data also for interfacial properties. A standard for storage and retrieval of such data has to be developed in future. Several thermophysical data such as density and thermal expansion coefficient can be extracted as temperature- and composition-dependant values from these databases both for pure phases and phase mixtures.

For the exchange of thermophysical material data along the process chain – stemming from experiments or a weak coupling between the length scales – a simple file-based format is used on the simulation platform. The term *material*, however, is somehow not quite unique and scale dependent. For *ab initio* simulations, the material data may describe the properties and the interaction of single atoms; in microscale simulations, the term material may be identified with the properties of the pure phase, which may be a mixture of several chemical species, whereas in macroscale process simulations it corresponds to the effective properties of a representative volume element.

The material data is stored in plain ASCII files. They can be created by the use of a text editor, data logging programs, or scale bridging tools such as homogenization or virtual testing software codes (Chapter 5). The file consists of lines containing key words and data. The data describes basically the material properties as tables of the temperature or other state variables. The following applies to all keyword and data lines of the material data file:

1) The first nonblank character of a keyword line must be a slash (/).
2) A line beginning with a hash (#) is a comment line and is ignored as well as any blank characters ( ).
3) A line may be maximum 256 characters long. There is no case sensitivity.
4) Parameters and options are separated by a comma (,). A keyword must also be followed by a comma, if any parameters are given.
5) If a parameter owns a value, the equal sign (=) is always used. The value may be a number or a character string.
6) A line ending with a comma (,) is continued in the next line.
7) Character strings as well as keywords may be 80 characters long.
8) All properties are defined in SI units.

The keywords for the naming of the data sets are usually the English standard names of the thermal, physical, mechanical, electrical, or optical property. The keywords already defined are given in Table 4.2. As mentioned before, an updated list of keywords can be found on the AixViPMaP site [1].

The data tables in the data section define, in general, piecewise linearized functions of the data values. The functions always depend on the temperature, and may also depend on other state variables. The type of interpolation scheme for the calculation of the values in between the supporting points is left to the software tool. The definition of the tables is as follows:

Table 4.2 Keywords for the naming of material data sets.

Keyword	**Description**
CONDUCTIVITY	Thermal conductivity
DENSITY	Density
DRIVING FORCE	Driving force of phase transformation
FRACTION SOLID	Volume fraction solid during solidification
FRACTION LIQUID	Volume fraction liquid during condensation
HEAT CAPACITY	Volume-specific heat capacity
HEAT OF FUSION	Volume-specific heat of fusion
HEAT OF MELTING	Volume-specific heat of meting
HEAT OF CONDENSATION	Volume-specific heat of condensation
HEAT OF EVAPORATION	Volume-specific heat of evaporation
ENTHALPY	Volume-specific total enthalpy
ELASTICITY	Hooke tensor
PINNING FORCE	Zener force
MOLAR VOLUME	Volume of a mole atoms
TEMPERATURE SOLIDUS	Solidus temperature
TEMPERATURE LIQUIDUS	Liquidus temperature
TEMPERATURE GASEOUS	Gaseous temperature
YIELD STRESS	Yield stress

- Each line defines the data values for one combination of the temperature and optional solution-dependent state variables.
- The first columns of values define the material properties. The number of properties depends on the grade of the material anisotropy.
- The last column always defines the temperature.
- The properties must be listed in ascending order of the state variables and the temperature.
- The columns are separated by a comma (,).

Thermophysical data are, in general, defined as volume-specific properties. The directional dependence of material properties is defined by the `TYPE` parameter that may have the values `ISOTROPY/ISO`, `ORTHOTROPY/ORTHO`, or general `ANISOTROPY/ANISO`. The sequence of the directional-dependent coefficients for second-order tensors is as follows:

- **Orthotropy**: 11, 22, and 33 direction,
- **General anisotropy**: 11, 12, 13, 21, 22, 23, 31, 32, 33.

For fourth-order tensors such as the Hooke tensor in elasticity, the sequence of the coefficients is given by:

- **Orthotropy**: 1111, 1122, 2222, 1133, 2233, 3333, 1212, 1313, 2323,
- **General anisotropy**: 1111, 1122, 2222, 1133, 2233, 3333, 1112, 2212, 3312, 1212, 1113, 2213, 3313, 1213, 1313, 1123, 2223, 3323, 1223, 1323, 2323.

The example dataset for the temperature-dependent density of generic low carbon steels is as follows:

```
/DENSITY
    7.849E-06,    0.0
    7.630E-06,  650.0
    7.650E-06,  700.0
    7.400E-06, 1300.0
    7.317E-06, 1434.0
    7.310E-06, 1440.0
    7.105E-06, 1500.0
    7.012E-06, 1510.0
    7.000E-06, 1525.0
```

The stress–strain curves of an aluminum alloy are defined as a function of the temperature and the equivalent plastic strain as follows:

```
/YIELD STRESS, TYPE=VON MISES
  179.58,   0.00,   0.00
  160.74,   0.00, 100.00
   99.50,   0.00, 200.00
  218.68,   0.02,   0.00
```

```
193.03,   0.02, 100.00
119.22,   0.02, 200.00
```

Here, the first column lists the yield stress for a given combination of the equivalent plastic strain and temperature. The second column defines the equivalent plastic strain, here 0.00 and 0.02, as the first solution-dependent state variable and the last column defines the temperature values 0.0, 100.0, and 200.0 as the second solution-dependent state variable. The values are ordered first by the plastic strain and then by the temperature. In this example, only a linear stress–strain curve is displayed. By repeating the block for each strain, the example could be extended to a more complicated situation.

4.4 Application Programming Interface

Besides the file-based exchange of discretized field variables and material data, the standardization of the programming interface is used to convey functional dependencies such as thermal or mechanical material models between the different software codes. These functions are coded as subroutines or functions using a high-level programming language and are called many times during a calculation run. On the basis of the modularized structure of the platform, as displayed in Figure 4.2, the subroutines may transport the functional dependencies along the *horizontal* and the *vertical* direction of the map. The horizontal axis links the different software codes keeping the same scale or the same level of detail. The material models coded in these functions thus may be reused by the different software codes. This enables, for example, the use of a consistent material model in all codes. On the vertical axis of Figure 4.2, the use of the programming interface

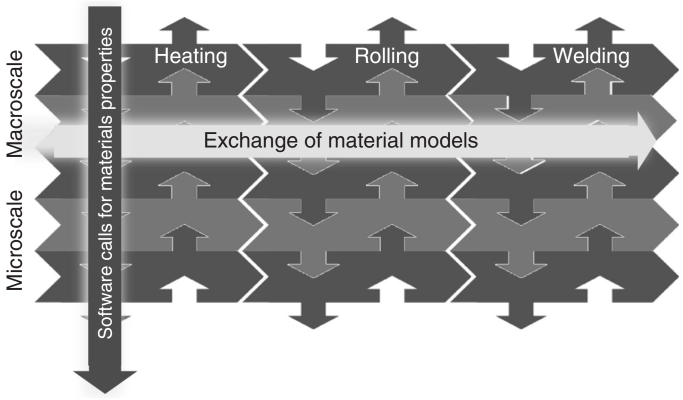
Macroscale
Microscale
Software calls for materials properties
Heating
Rolling
Welding
Exchange of material models

Figure 4.2 User programming interfaces for the exchange of material models between the software codes of the simulation chain, and the calculation of effective properties from smaller length scales.

may enable a direct link between the software codes at different length scales. Thus, it enables a strong coupling between the simulations on different length scales and the use of multiscale material models. This is especially helpful to realize multiscale material models using the multi level finite element analysis method or the numerical calculation of effective material properties based on discretized microstructures originating from transient phase-field models, virtual models such as DIGIMAT, or discretized metallurgical images. In principle, the call of any external software tool is possible as the subroutine itself may write the input to files and call any program by the use of a system call and read the results from file after the program run again.

The programming interfaces, which are currently already standardized, interact at different levels with the simulation software code and are very similar to programming interfaces already being implemented in several existing software codes such as ABAQUS, ANSYS, or FEAP. The standardized programming interfaces for the user subroutines are

- **USER_MATERIAL_TM**: user-defined mechanical constitutive material models,
- **USER_MATERIAL_HT**: user-defined thermal material laws,
- **USER_EXPANSION**: user-defined models for stress-free strains stemming from thermal dilatation or the volume change due to phase transformations,
- **USER_PHASE_TRANSFORMATION**: user-defined models for the description of phase transformations.

The definition of the programming interface comprises the definition of a fixed formal name of the subroutine and an argument list. The argument list defines the type, size, and content of the variables exchanged between the calling program and user subroutine. The variables are classified into three groups:

- Variables to be defined by the user. This is the required information for the calling program.
- Variables passed to the subroutine for information purpose. These variables should not be changed, and it is advised that the calling program passes a copy of the variable to the subroutine.
- Variables to be updated.

All subroutines are evaluated at each integration point of the discretized geometry. The subroutines may be called several times during an iterative solution procedure for the resolution of a geometric or material nonlinearity. The information on the iteration step and the calling sequence during the solution procedure is passed to the subroutine by several flags described in the following text.

The communication between the user subroutines is realized with the help of an integer and a real array. These arrays are used to store the user-defined state variables at each integration point. The calling program must ensure to pass the correct array of state variables to the call of the subroutine at each integration point. The size and the use of the state variables must be defined by the user. Subroutines for the initialization and the output of the variables for postprocessing are not standardized, and the features are left to the software provider.

4.4.1 USER_MATERIAL_TM Subroutine

The `USER_MATERIAL_TM` subroutine is used to implement a user-defined material model into an existing software code. The subroutine may be exchanged by the different software tools being used for the simulation of a process chain at a given length scale. For multiscale material models, the subroutine may consist of a complete calculation tool or a system call that, for example, passes the effective tangential material stiffness to the calling program at a larger length scale.

The formal programming interface in FORTAN notation of the subroutine is given by:

```
 SUBROUTINE USER_MATERIAL_TM (STRESS,DSTRSEPS,DSTRDTEMP,ENERGY,
1                             STRESS,STATEV,NSTATV,STRAIN,
2                             DSTRAIN,TIME,DTIME,TEMP,DTEMP,
3                             NAME,NTENS,PROPS,NPROPS,COORDS,
4                             ISTEP,IITER,IELEM,IGAUSS,ISTATUS)
CHARACTER   NAME(80)
REAL              STRESS(NTENS),STATEV(NSTATV),ENERGY(2,2),
1                 DSTRDEPS(NTENS,NTENS),DSTRDTEMP(NTENS),
2                 STRAIN(NTENS),DSTRAIN(NTENS),STRESS(NTENS),
3                 PROPS(NPROPS),COORDS(3),TIME,DTEIM,TEMP,DTEMP
INTEGER           NTENS,NSTATV,NPROPS,ISTEP,IITER,IELEM,IGAUSS,
1                 ISTATUS
user coding to define DSTRDEPS, STRESS, STATEV
and, if necessary, ENERGY, DEPSDTEMP
The solution dependent state variables may be updated

END
```

A number of routines for simple linear elasticity written in FORTRAN and C can be found on the AixViPMaP web site [1] for free download. In the following text, the variables are described in detail. The ordering of the directions of the stresses and strains is as follows: first, the direct components are ordered by the direction followed by the shear directions again ordered after their directions.

Variables Required by the Program

`DSTRDEPS(NTENS,NTENS)`	The algorithmic tangent matrix of the constitutive model with respect to the strains. The variable `DSTRDEPS(I,J)` defines the change of the Ith stress component caused by an infinitesimal change of the `J`th component of the strain tensor. Remember, the calling program may accept only the symmetric part of the array.

`DSTRDTEMP(NTENS)`	The algorithmic tangent matrix of the constitutive material model with respect to the temperature. Here, `DSTRDTEMP(I)` defines the change of the Ith stress component caused by an infinitesimal temperature change.
`STRESS(NTENS)`	The array contains the Cauchy stress tensor at the beginning of the increment and must be updated in this routine to be the stress tensor to the end of the increment.
`ENERGY(2,2)`	The strain energy at the end of the increment and its derivation with respect to the strain. `ENERGY(1,1)` is the reversible part of the strain energy, `ENERGY(2,1)` is the irreversible part of the strain energy, and `ENERGY(1,2)and ENERGY(1,2)` contain the corresponding linearized deviations.

Variables to Be Updated

`STATEV(NSTATV)`	The array contains the solution-dependent state variables. They correspond to the values at the beginning of the increment, unless they are updated by other user-defined subroutines. The values passed correspond only to the current integration point for which the subroutine is passed and they may be used for data exchange between the user subroutines. The memory allocation is left to the calling program. In addition, there should be the possibility in the calling program to initialize and output the variables.

Variables Passed in for Information

`STRAIN(NTENS)`	The array contains the total strains at the beginning of the increment, for example, the equilibrium state of the previous step. The definition of the strain is left to the calling program. For compatibility reasons, the use of the Lagrangian strain in large deformation analysis should be preferred. For coupled thermomechanical problems, this is the stress-generating part of the strain only. Other stress-free strains such as thermal strains and dilatations due to phase transformations are calculated by the UEXPAN subroutine and have already been subtracted.
`DSTRAIN(NTENS)`	The array of strain increments within this load step. As before, these are the mechanical parts of the strain only.
`TIME`	The simulation time information at the beginning of the increment.
`DTIME`	The time period covered by the current increment.
`TEMP`	The temperature at the integration point at the beginning of the current increment.
`DTEMP`	The temperature change during the previous time increment.

NAME	The user-defined name of the material region, left justified.
NTENS	Number of the stress or strain components.
NSTATV	Number of solution-dependent state variables that are associated with this material model.
PROPS (NPROPS)	User-specified array of material constants associated with this user material as defined in the input file of the calling program.
NPROPS	User-defined number of material constants.
COORDS (3)	The current coordinates of this integration point.
ISTEP	Number of the current time step.
IITER	Current iteration number.
IELEM	Number of the element.
IGAUSS	Number of the integration point within the element.
ISTATUS	Flag indicating if the routine is called at the beginning of the increment (ISTATUS=1), at the end of the increment (ISTATUS=2), or elsewhere (ISTATUS=0).

4.4.2 USER_MATERIAL_HT Subroutine

The thermal counterpart of the previously described subroutine is the USER_MATERIAL_HT programming interface. It is used to implement a thermal material model with a complex thermal behavior including the release or consumption of the heat of fusion, heat of evaporation, heat of solidification, heat of melting, or the heat release during solid-state transformations. The programming interface is a subclass of the user–material programming interface. Thus, it should provide the same user-defined variables and should be called at the same integration points as all other subroutines. The header and input/output variables of the USER_MATERIAL_HT routine expected from the driver routine are as follows:

```
SUBROUTINE USER_MATERIAL_HT (ENTHALPY,DENTHALPYDT,CONDUCTIVITY,
1                               STATEV,NSTATV,TIME,DTIME,NAME,
2                               PROPS,NPROPS,COORDS,NDIM,ISTEP,
3                               IITER,IELEM,IGAUSS,ISTATUS)
CHARACTER     NAME(80)
REAL          ENTHALPY,DENTHALPYDT,CONDUCTIVITY(3,3),
1             CONDUCTIVITY(3,3),STATEV(NSTATV),PROPS(NPROPS),
2             COORDS(3)
INTEGER       NDIM,NSTATV,NPROPS,ISTEP,IITER,IELEM,IGAUSS,
1             ISTATUS
user coding to define ENTHALPY,DENTHALPYDT and CONDUCTIVITY

END
```

Variables Required by the Program

`ENTHALPHY`	The total volume specific enthalpy at the integration point.
`DENTHALPHYDT`	The linearized derivation with respect to the temperature at the integration point.
`CONDUCTIVITY(3,3)`	The current conductivity of the material at the integration point.
`DCONDUCTIVITYDT(3,3)`	The deviation of the current conductivity of the material at the integration point with respect to the temperature.

Variables to Be Updated

`STATEV (NSTATV)`	The array contains the solution-dependent state variables. They correspond to the values at the beginning of the increment, unless they are updated by other user-defined subroutines. The values passed correspond only to the current integration point for which the subroutine is passed, and they may be used for data exchange between the user subroutines. The memory allocation is left to the calling program. In addition, there should be the possibility in the calling program to initialize and output the variables.

Variables Passed in for Information

`NDIM`	The number of coordinates indicating a 2D or 3D analysis.

The description of the remaining variables passed for information is given in detail in the section titled Variables Passed in for Information.

4.4.3
USER_EXPANSION Subroutine

The programming interface of the user expansion subroutine must be considered as a subset of the programming interface for user material models. The subroutine can be used to implement stress-free eigenstrains stemming from thermal dilatations or from density changes caused by phase transformations. The implementation of the programming interface should enable the use of the solution-dependent state variables as used in the USER_MATERIAL_TM subroutine in order to allow the realization of complex models for phase transformations. The subroutine should be called at the same integration points, as the other user subroutines. The ordering of the calling sequence of the several subroutines is left to the calling program. The formal interface of the user subroutine is as follows:

```
SUBROUTINE USER_EXPANSION   (EXPANSION,DEXPANDTEMP,STATEV,
1                           NSTATV,TIME,DTIME,TEMP,DTEMP,
2                           PROPS,NPROPS,NAME,COORDS,NTENS,
3                           ISTEP,IITER,IELEM,IGAUSS,ISTATUS)
CHARACTER  NAME(80)
REAL            EXPAN(NTENS),DEXPANDT(3),PROPS(NPROPS),
1               STATEV(NSTATV),COORDS(3),TIME,DTIME,TEMP,
2               DTEMP
INTEGER         NTENS,NSTATV,NPROPS,ISTEP,IITER,IELEM,IGAUSS,

1               ISTATUS
user coding to define EXPAN, DEXPANDT and update STATEV if nec-
essary.

END
```

Variables Required by the Program

EXPANSION (NTENS)	The tensor of the total stress-free strains for the current configuration. The ordering of the components of the tensor in Voigt notation coincident with the stresses and strains, in general.
DEXPANDTEMP (NTENS)	The linearized derivation of the stress-free strains with respect to the temperature.

Variables to Be Updated

STATEV (NSTATV)	The array that contains the solution-dependent state variables as described before.

Variables Passed in for Information The description of the variables passed for information is given in detail in **Variables Passed in for Information**.

4.4.4 USER_PHASE_CHANGE Subroutine

This programming interface allows the definition of a user-defined material model for the solidification and solid-state transformations. It consists basically of two interfaces: the interface to the user material subroutine, which has to deliver the thermal and mechanical material model, and the interface to the user expansion subroutine, which is realized as a subclass of the user material interface. The user expansion subroutine takes care of the density changes and eigenstrains during the solid-state transformation and includes them into the calculation of the mechanical submodel. The communication between all subroutines should be realized by the usage of the same fields of the user-defined state variables. All subroutines should have access to the same variable space and should be called at the same integration points. This subroutine is intended to be a generalized interface to manipulate the phase fractions of the several metallurgical phases in a multiphase analysis, to interact with the calling software code in a more detailed way, and to allow the user to change the chemical composition and others. Usually, the volume or mass fractions of the metallurgical phases are stored as user-defined state variables. Thus, the phase changes may be simply modeled internally in the USER_MATERIAL_HT subroutine or in more detail using the subroutine described here.

The header and input/output variables of the USOLID routine expected from the driver routine are as follows:

```
SUBROUTINE USER_PHASE_CHANGE     (ENTHALPY,DENTHALPYDT,
1                                 DMASS,DRIVINGFORCE,
2                               SPECIES,DSPECIES,NSPECIES,
3                                 PFRAC,DPFRAC,NPFRAC,
4                                 STATEV,NSTATV,TIME,DTIME,
5                                 TEMP,DTEMP,PROPS,NPROPS,
6                                 NAME,COORDS,NDIM,ISTEP,
7                                 IITER,IELEM,IGAUSS,ISTATUS)

CHARACTER  NAME(80)
REAL            ENTHALPY,DENTHALPYDT,DMASS(NPHASES),
1               DRIVINGFORCE(NPFRAC,NPFRAC),
2               SPECIES(NPFRAC,NSPECIES),
3               DSPECIES(NPFRAC,NSPECIES),
4               PFRAC(NPFRAC),DPFRAC(NPFRAC),STATEV(NSTATV),
5               TIME,DTIME,TEMP,DTEMP,PROPS(NPROPS),COORDS(3)
INTEGER         NDIM,NSTATV,NPROPS,ISTEP,IITER,IELEM,IGAUSS,
1               NSPECIES,NPFRAC,ISTATUS

user coding to define ENTHALPY, DENTHALPYDT, DMASS, DRIVINGFORCE,
SPECIES, DSPECIES, PFRAC or DPFRAC. All variables are optional and
depend largely on the calling software code

END
```

Variables Required by the Program

ENTHALPHY	The total volume-specific enthalpy at the integration point.
DENTHALPHYDT	The linearized derivation with respect to the temperature at the integration point.

`DMASS (NPFRAC)`	The increment of exchanged mass between the phases during the last time step. To fulfill the balance of mass, the sum of the exchange quantities should vanish.
`DRIVINGFORCE (NPFRAC,NPFRAC)`	The forces driving the transformation between the several phases.
`SPECIES (NPFRAC,NSPECIES)`	Chemical composition of the phases.
`DSPECIES (NPFRAC,NSPECIES)`	Change of the chemical composition of the phases.
`PFRAC (NPFRAC)`	Volume fraction of the distinguished metallurgical phases.
`DPFRAC (NPFRAC)`	Change in the volume fraction of the distinguished metallurgical phases.

Variables to Be Updated

`STATEV (NSTATV)`	The array contains the solution-dependent state variables. They correspond to the values at the beginning of the increment, unless they are updated by other user-defined subroutines. The values passed correspond only to the current integration point for which the subroutine is passed, and they may be used for data exchange between the user subroutines. The memory allocation is left to the calling program. In addition, there should be the possibility in the calling program to initialize and output the variables.

Variables Passed in for Information

`NDIM`	The number of coordinates indicating a 2D or 3D analysis.
`NSPECIES`	Number of the chemical species for each metallurgical phase.
`NPFRAC`	Number of phases.

The description of the remaining variables passed for information is given in detail in **Variables Passed in for Information**.

4.5 Future Directions of Standardization

The above standardization scheme has been implemented in a number of software codes available at the RWTH Aachen University, and converters have been programmed to transfer results of some commercial codes into this standard, which by now essentially covers thermodynamic and thermomechanical aspects. The combination of different simulation tools – both academic and commercial – has allowed for the successful simulation of some test scenarios (Chapters 8–12). It should be noted that the standardized data exchange between the different models drastically reduced the load of data conversion, a significant effort before introducing the standard. Further development of this standard has to be discussed within the emerging ICME community and to proceed between users of ICME concepts and software developers – academic and/or commercial – providing respective models and tools.

References

1. *http://www.aixvipmap.de.*
2. *http://www.vtk.org.*
3. *http://www.paraview.org.*
4. *http://www.vtk.org/pdf/file-formats.pdf.*