

A FRAMEWORK FOR A VIRTUAL MATERIAL
TESTING LABORATORY

A FRAMEWORK FOR A VIRTUAL MATERIAL
TESTING LABORATORY

By
HUANCHUN GAO, M.ENG B.ENG

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree
Master of Applied Science

McMaster University

©Copyright by HuanChun Gao, March 2004

MASTER OF APPLIED SCIENCE (2004) McMaster University
(Software Engineering) Hamilton, Ontario

TITLE: A Framework for A Virtual Material Testing Laboratory

AUTHOR: HuanChun Gao, M.Eng B.Eng (JiLin University of Technology)

SUPERVISOR: Dr. Spencer Smith

NUMBER OF PAGES: xii, 261

Abstract

This thesis presents a framework for a virtual laboratory for material testing, called *Virlab*. A virtual laboratory is an open and flexible environment that is used to simulate a set of experiments using a computer. It is beneficial and valuable for researcher and educators to simulate real problems and to conquer some challenges such as a weightless body. The virtual laboratory for material testing contributes both to the field of mechanics of materials and the field of software engineering.

In the field of material mechanics *Virlab* can be used for material testing education and research. Students can rapidly investigate many experiments for materials and the difference between kinematics quantities and stress measures. *Virlab* also offers a convenient platform for researchers to investigate and test new constitutive equation and implement their new algorithms. *Virlab* also encourages unambiguous definitions of mechanics terms and principles.

In the field of software engineering the contribution is to provide an example of the application of software engineering approaches to an important scientific computing problem. By showing the successful application of software engineering methodologies for a virtual laboratory, it is hoped that software engineering ideas will spread to other scientific applications. In terms of software engineering methodologies, this thesis presents a component-based design for the virtual laboratory for material testing. In this thesis we conduct a commonality analysis for material testing, decompose the system into modules with the information hiding

principle, provide an easy way to identify components from the module decomposition, and build the component-based system architecture. In this procedure we apply the concept of design through documentation at each stage.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Dr. Spencer Smith, for the considerable effort he exerted on my behalf throughout the research of this thesis, and for providing a good example for a dedicated professor. After I had my son during my graduate studies, I and my husband named my son Spencer and hope he would be a nice, smart and dedicated person like Dr. Spencer Smith.

Thoughtful comments from Dr. Alan Wassyng and Dr. Antoine Deza have helped to clarify the ideas expressed in this work.

I appreciate very much the financial support from my supervisor and the Department of Computing and Software.

I am grateful to my Mom and Dad for their care, especially they gave me strong supports after I had my son so I can concentrate on my research.

Finally, I would like to thank my husband Wenli and my son Spencer for their love and encouragement.

Table of Contents

Abstract	iii
Acknowledgements	v
Table of Contents	vi
List of Figures	vii
List of Tables	ix
List of Symbols	x
1 Introduction	1
1.1 What is a virtual laboratory?	4
1.2 Why is a virtual laboratory needed?	5
1.3 An overview of real material testing	7
1.4 Thesis motivation	10
1.5 Thesis scope	12
1.6 Thesis organization	13
2 Commonality analysis of material experiments	15
2.1 Terminology	16
2.2 Commonalities	25
2.3 A case study: a uniaxial displacement-controlled experiment for a viscoelastic material	30
2.4 Conclusions	37
3 Component-based design	44
3.1 What is a component?	44
3.2 Why use component-based system design?	47
3.3 Modularity	51

3.3.1	System decomposition	51
3.3.2	Hierarchical structure	57
3.4	Identifying the components	63
3.5	Component-based system architecture	68
3.6	Documentation	71
3.6.1	Documenting modules	72
3.6.2	Documenting the module interface specification	75
3.6.3	Documenting the component description	84
4	An overview of the <i>Virlab</i> software	90
4.1	An introduction to the <i>Virlab</i> software	91
4.1.1	The main window	92
4.1.2	The setup window	94
4.1.3	The output window	95
4.2	An example experiment in <i>Virlab</i> for a uniaxial displacement-controlled experiment for a viscoelastic material	99
5	Conclusions, contributions and future work	106
5.1	Conclusions	106
5.2	Contributions	108
5.3	Future Work	111
	Bibliography	114
A	Relationship among kinematics quantities	120
B	The solution for the uniaxial extension displacement-controlled experiment	124
C	The module guide for <i>Virlab</i>	127
D	The module interface specification for <i>Virlab</i>	147
E	Component description for <i>Virlab</i>	254
F	The procedure for adding a new constitutive equation	259

List of Figures

1.1	A tensile testing machine from the Instron Corporation	9
1.2	Schematic illustration of the tensile machine shown in Figure 1.1	10
2.1	Coordinate system [Maz70]	17
2.2	A displacement-controlled experiment	19
2.3	A load-controlled experiment	19
2.4	A biaxial experiment	20
2.5	A multiaxial experiment	21
2.6	Elongation of a rod subjected to a uniaxial tensile force	23
2.7	A multiaxial experiment for the commonality analysis	27
2.8	A uniaxial displacement-controlled experiment	30
2.9	Comparison of data for the correct and incorrect experiment	33
3.1	Module hierarchy	53
3.2	Use relation in the <i>kinematics module</i>	59
3.3	Use relation in the <i>stress module</i>	60
3.4	Use relation of the system	61
3.5	The data flow chart for an experiment	62
3.6	System architectural layer	65
3.7	Component architecture of the <i>Virlab</i> system	70
4.1	The window for the experiment type selection	93
4.2	Experiment window	94
4.3	Setup window	95

4.4	The SpecifySpecimenGeometry window	96
4.5	The SpecifyTimeSetup window	96
4.6	The ConstitutiveEquation window	97
4.7	The SpecifyLoadOrDisplacement window	97
4.8	The output window	98
4.9	The selection of a uniaxial displacement-controlled experiment . .	99
4.10	Experiment window for a uniaxial displacement-controlled experi- ment	100
4.11	Input the function information	100
4.12	Input the geometry information	101
4.13	Specify the constitutive equation	102
4.14	Specify the time configuration	102
4.15	Select the function type for displacement variable	103
4.16	The output for the experiment	104

List of Tables

2.1	A comparison of different material models	26
2.2	The specification of the displacement function in the uniaxial experiment	31
2.3	The specification of Maxwell's equation	31
2.4	Assumed parameters for the Maxwell's equation	32
2.5	Visualizing the <i>Virlab</i> system	38
2.6	The list of unlikely changes	40
2.7	The list of anticipated changes	43
3.1	The comparison between the five layers in the <i>Virlab</i>	67
3.2	Component categories in the <i>Virlab</i>	68
3.3	<i>Experiment definition module</i>	73
3.4	The order for the initialization	80
3.5	An example for an event table	82
3.6	A component description for the kinematics component	87
3.7	Output for <code>kc_knownquantity</code>	88
3.8	Output for <code>kc_geometry</code>	88

List of Symbols

$ox_1x_2x_3$	Eulerian coordinate system
x_1, x_2, x_3	Axes in Eulerian coordinate system
$OX_1X_2X_3$	Lagrangian coordinate system
X_1, X_2, X_3	Axes in Lagrangian coordinate system
t	Time
Δt	Time step
L_0, W_0, H_0	The original length, width and height of a test specimen, respectively
L, W, H	The updated length, width and height of a test specimen, respectively
u, v, w or u_1, u_2, u_3	The displacements of the test specimen in the x_1, x_2, x_3 axial direction, respectively
$\bar{u}_1, \bar{u}_2, \bar{u}_3$	The displacements at the end of the test specimen in the x_1, x_2, x_3 axial direction, respectively
F	Load
A_0	The original cross-sectional area of the specimen
A	The updated cross-sectional area of the specimen
σ	Stress
ε	Strain

σ_x	The stress in the x_1 direction
ε_x	The strain in the x_1 direction
ε_y	The strain in the x_2 direction
ε_z	The strain in the x_3 direction
ν	Poisson's ratio
$\dot{\sigma}$	The rate of the change of the stress
$\dot{\varepsilon}$	The rate of the change of the strain
f	A load function of time
\dot{u}	A velocity function of time
$\vec{f} = (f_1, f_2, f_3)$	A vector representing the forces applied to the sides of the test specimen
$\vec{u} = (\bar{u}_1, \bar{u}_2, \bar{u}_3)$	A vector representing the displacement at the end of the specimen
$\vec{u}(t)$	The displacement function
$\dot{\vec{u}}(t)$	A velocity function of time in the displacement-controlled experiment
$\ddot{\vec{u}}(t)$	An acceleration function of time in the displacement-controlled experiment
$\vec{f}(t)$	A force function of time
$\dot{\vec{f}}(t)$	The first time derivative of force in the load-controlled experiment
$\ddot{\vec{f}}(t)$	The second time derivative of force
λ	Relaxation time
η	Viscosity
E	Young's Modulus or elastic modulus
τ	The stress in the description of a viscous material
D	The rate of deformation in the description of a viscous material

Chapter 1

Introduction

Scientists and engineers face diverse and complex challenges when designing and developing advanced experiments. For example, a tensile test is challenging in part because of the wide range of materials that are tested, such as linear elastic materials, hyperelastic materials, hypoelastic materials and viscoplastic materials *etc.* Another challenge in material testing is how to describe different materials. For some materials, such as linearly elastic materials, the description is a simple linear state equation relating stress and strain, but for other materials, such as elasto-viscoplastic materials, the description is complex with several nonlinear equations, multiple parameters and a nontrivial relationship between the stress and the deformation history.

A significant source of problems with the real experiment is ambiguous definitions of the requirements. In material testing scientists and engineers often struggle to find ways to describe the materials and the material tests in an unambiguous way. One problem, for instance, is the definition of strain. Should it be defined relative to the new or the old configuration, or to some other configuration? The interpretation of the experimental results in terms of a material model

requires a clear definition. The solution to this problem is to use mathematical techniques and notations to unambiguously specify the requirements. This means a formal approach, which can be supported by a virtual laboratory. A virtual laboratory is a software environment that simulates real world experiments. Ideally, a virtual laboratory will unambiguously present all the required definitions and calculate values for the quantities of interest, such as stress and strain. The users will be able to use this information to determine the best material model for their situation. Moreover, a virtual laboratory has the advantage that it can force the users to think about the inappropriateness of mixing incompatible quantities in their material models.

Another source of problems in material testing is the technical difficulty of setting-up an experimental environment that matches the assumptions used to model the experiment. For example, the theoretical model of many experiments ignores the effect of the material's self-weight. In a real experiment it is impossible to have a weightless material, but in the virtual laboratory it is easy to simulate a weightless specimen and moreover, the set-up of the experiment is simple as it only requires entering some data and clicking a few buttons.

Not only researchers and engineers are confronted with challenges. Educators also have faced difficulties when teaching students the rules of mechanics. If the teachers had access to a frictionless environment, this would simplify teaching students such principles as Newton's laws of motion. It is impossible however, to set up a perfect experimental environment where there is no friction. However, in a virtual laboratory it is easy to model a friction free environment.

Some difficulties from real experiments can be conquered by sacrificing considerable time and expense, while others can never be overcome. These difficulties together with the needs of scientific discovery motivate researchers to build virtual laboratories. Ongoing work in the area of virtual laboratories is mostly focused on research on different problems in a variety of fields. For instance, people who work on education and training contribute to virtual laboratories for education. One area of contribution is related to the remote education based on the World Wide Web [KCZ+01][GAP+02][Bud01][Sch99]. Other contributions from the literature are related to virtual laboratories for different disciplines. Some examples of virtual laboratories include, a virtual geotechnical laboratory for soil tests and triaxial tests [PZF00], a virtual chemical lab for inorganic chemical reactions [REG+00], a virtual lab for mechanics and materials science [KJR02] [WSS90] and a virtual laboratory for biology [Mer91].

The concept of a virtual laboratory changes as the field of application changes. In the area of education, the virtual laboratory can be imagined to verify the mechanics rules at work by performing a virtual physical experiment [Web_mec_02] [Web_Jhu_02] [Web_phy_02]. Sometimes virtual laboratories for education have additional features, such as a search engine to look for the related information from a textbook [Web_mec_02]. Virtual laboratories also provide an environment to analyze experimental data and output the result data [KJR02][WSS90]. In other areas such as aerospace engineering, deep ocean engineering and nuclear engineering, virtual laboratories are used as a simulator to study dangerous situations [AKB+]. Given the range of application of the term virtual laboratory, the question remains what is a virtual laboratory? This question is addressed in

this chapter.

Section 1.1 gives the definition of a virtual laboratory. Section 1.2 describes the reasons that a virtual laboratory is needed. As the virtual lab of interest in our current studies will focus on material testing, Section 1.3 provides an overview of real material testing. Section 1.4 summarizes the difficulties from real material testing and gives the motivations for the virtual laboratory. Section 1.5 discusses the scope for the virtual laboratory system and this thesis. Finally, Section 1.6 provides an overview of the thesis.

1.1 What is a virtual laboratory?

Although some virtual laboratories are limited in the sense that they provide a solution to certain specific problems in a given field, some commonalities exist among virtual laboratories. The most obvious commonality is that they provide a software environment. Therefore, in this thesis a virtual laboratory is defined as an open and flexible software environment that is used to simulate a set of experiments using a computer. In the virtual laboratory each virtual experiment is added in a similar sense as adding new equipment and experimental methods to a real laboratory.

A significant benefit of giving a definition of the term virtual laboratory is that researchers, engineers and educators need a common definition if they are to communicate effectively about the idea of a virtual laboratory.

1.2 Why is a virtual laboratory needed?

Most currently existing virtual laboratories are focused on educational purposes either for conducting detailed experiments, such as physics experiments, or for implementing distance education on the World Wide Web. There are also a few virtual laboratories that emphasize scientific research on a specific domain. However, sometimes a good educational virtual laboratory can also be used for research. For example, a virtual laboratory [KCZ+01] for control experiments on a coupled tank apparatus is being utilized in the teaching of undergraduate courses. In this virtual laboratory students learn to identify a physical model for the coupled tank system based on input-output data and design a PID controller and different fuzzy logic controllers for the system. Postgraduates also use this virtual laboratory to solve both classical optimal control system design problems and advanced robust control problems. This virtual laboratory also offers an excellent and convenient platform for researchers to test and implement their new algorithms. No matter the purpose of a virtual laboratory, the reasons that a virtual laboratory is needed are as follows:

- Budget

Building a real laboratory with a sufficient number of adequate training and/or research experiments may be very expensive. However in a virtual lab, a set of specific experiments are inexpensively integrated in the software. The cost of a virtual laboratory is much lower than for a real laboratory.

- Space

A real laboratory needs enough space to accommodate the equipment and

people who will use the laboratory. The concept of a virtual laboratory requires no real space, other than space for the computer equipment.

- Time

To researchers, it is time consuming to set up a scientific research laboratory. However in a well-designed virtual laboratory, it is easy to simulate scientific problems. To students, a virtual laboratory based on WWW environment means that students have a flexible lab time to finish their experiments by choosing to do experiments at home or at school. In particular, a virtual laboratory is more convenient for part-time students holding full-time employment in industry.

- Hands-on experiences in what-if scenarios

In a virtual laboratory, students can do experiments under a variety of conditions, including extreme conditions. This experience will benefit them when they take part in designing for industry. Furthermore, researchers on theoretical analysis and numerical simulation can consider building novel and highly experimental designs, without the risk of actually physically building them. Furthermore, a virtual laboratory [KCZ+01] provides a convenient platform for researchers to test and implement their new algorithms.

- Danger

Some experiments such as an underwater experiment or a nuclear experiment are dangerous. A minor error might result in loss of life. However in the virtual laboratory detailed experiments can be done exhaustively. If a real experiment is still necessary, the preliminary virtual experiment can greatly

decrease the possibility of an accident.

- Political, cultural or governmental consideration

Sometimes because of political, cultural or governmental reasons, some real experiments are not allowed to be conducted. As an example, the Sandia National Laboratories are required to do all of their nuclear experiments with computer simulation, as the U.S. government has banned actual nuclear testing [Web_Sandia_03]. The only option for the lab to continue their research is to use a numerical simulation.

- Technical difficulties

In real experiments, it is impossible to get a perfect environment that exactly matches the model being used, but in the virtual laboratory it is easy to set up a perfect environment. For instance, in the virtual laboratory it is easy to build an environment without friction, but in a real experiment some friction is always present. Weightless parts are also possible in a virtual laboratory.

The above benefits provide the justification for the virtual approach when considering research on materials testing. Before discussing a virtual materials testing laboratory, we need to review the test that are done in a real laboratory.

1.3 An overview of real material testing

This section is intended to generally describe material testing. The in-depth details and terminology of material testing will be explained in Chapter 2. To

give an overview of material testing, we have to mention a few necessary terms, which will be written in italics.

In industry, important factors for the improvement in the performance and reliability of products are the development of new materials, the novel use of existing materials, better understanding of the structure-property relationships and incorporation of both mechanics and material science in the design of structures [KJR02]. It is obvious that correct understanding of materials properties is necessary, regardless of whether new materials or existing materials are being considered. A correct understanding of materials comes from exhaustive testing of materials. As far as testing is concerned, there are many tests, such as engineering tensile testing, hardness testing, ductility testing, shear testing and so on. Different tests are used to determine different material properties. For example, in an engineering tensile test the material properties that can be obtained are as follows: modulus of elasticity, yield strength, ultimate tensile strength, percent elongation at fracture and percent reduction in area at fracture [Smi93].

Since material testing is a broad and deep topic, we have to restrict our focus. In material testing we are interested in *kinematics quantities*, *stress* and *constitutive equations*. *Kinematics quantities* are used to describe the deformation of the material in response to the external forces acting on the material body. *Stress* is the internal resistance in the material to external forces or reactions acting on the material [Pet69]. *Constitutive equations* are a mathematical relationship among internal attributes of the material and describe the possible deformation history dependent relationship between stress and kinematics quantities. In material testing we want to precisely specify the constitutive equations,

obtain experimental data such as kinematics quantities and stress, plot the data, and analyze the data to obtain the relationship between kinematics quantities and stress. In the real laboratory the materials tests are conducted by professional material testing equipment.

Figure 1.1 shows a picture of a modern tensile testing machine [Smi93]. The force (load) on the test specimen is recorded on the chart paper in the drawer on the left. The *strain*, which describes the deformation in the dimension or shape of the material [BJ81] that the test specimen undergoes, is also recorded on the chart. The signal for the strain is obtained from the extensometer attached to the test specimen.

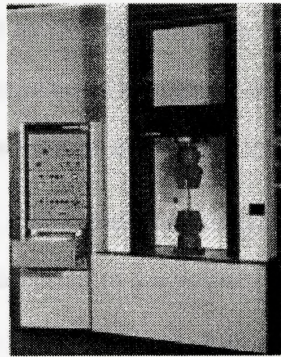


Figure 1.1: A tensile testing machine from the Instron Corporation

Figure 1.2 illustrates schematically how the test specimen is tested in tension [Smi93]. In this figure the external load or displacement on the moving crosshead causes the test specimen to deform. The deformation information is helpful to

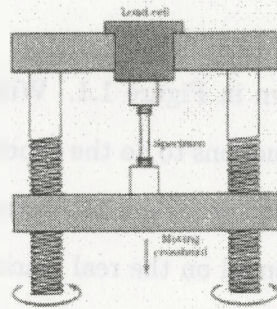


Figure 1.2: Schematic illustration of the tensile machine shown in Figure 1.1

research the material. There are two classes of experiments: *load-controlled experiments* and *displacement-controlled experiments*. In the load-controlled experiment, the test specimen is extended or compressed by an applied load and the load is the independent variable. Compared with the load-controlled experiment the test specimen in the displacement-controlled experiment is extended or compressed by a displacement and the displacement is the independent variable.

1.4 Thesis motivation

After an overview of real material testing was introduced in Section 1.3, some difficulties of real material testing become clear.

- Hard to overcome technical difficulties

The weight of the test specimen and constraints on the boundary conditions cannot be ignored. They are involved in the test and make the analysis of

the test more complicated. Also precise control of the deformation velocity or load can be challenging.

- Limited experiments

A real test machine is shown in Figure 1.1. With a real machine, people should follow the fixed instructions to do the experiment. It is impossible to do exhaustive experiments and provide what-if scenarios because some functions might not be implemented on the real machine. Limited experiments are not always enough for a researcher to completely test and understand a new material.

- The considerations of space, money and time

To do a real material test on the corresponding test machine, we need money for the machine, space to place the machine and we need considerable time to set up the experimental environment.

The above technical difficulties, financial requirements and operational inconveniences motivate us to develop a virtual laboratory for material testing. In the virtual laboratory, it is easy to set up perfect boundaries, implement a weightless test body and control the deformation velocity, implement and test new constitutive theories and algorithms. Our purpose is not to provide anything that cannot be done by a general purpose simulation package. Our focus is that the system will provide benefits for researchers and educators. Additionally during our research on the virtual laboratory we found that there are few existing virtual laboratories designed for material testing, except for a tensile test that was implemented in [KJR02]. This deficiency motivates us to develop a virtual laboratory for material

testing, that is not for testing real material but for providing a software environment to simulate a set of experiments.

From the papers describing existing virtual laboratories we found that existing virtual laboratories were seldom designed with a software engineering approach. For instance, the systems are not developed using modular decomposition, component composition, or the corresponding documentation system. This deficiency in the research of material testing obviously indicates a gap between the mechanics field and software engineering field. The developers and engineers in the field of mechanics lack the knowledge of software engineering approaches and the developers in the field of software engineering have not applied their approaches to the practical application of material testing. This gap also motivates us to present an example on how to apply software engineering approaches to design a virtual laboratory for material testing, which is named *Virllab*.

1.5 Thesis scope

Having identified the difficulties for real material tests, we propose a virtual laboratory for material testing. However, it is unrealistic that a virtual laboratory can be quickly developed for all material tests. So first we have to restrict the scope for the experiments in the *Virllab*.

The experiments that can be done in the virtual laboratory will be limited to the displacement-controlled experiments and load-controlled experiments. In each of these types, the experiment can be a *uniaxial experiment*, which is conducted just from one axial direction, a *biaxial experiment*, which is conducted from two

axial directions, or a *multiaxial experiment*, which is conducted for all three axial directions. In all experiments, we are after kinematics quantities, stresses and the relationship between kinematics quantities and stress through using a precisely specified constitutive equation.

Our focus is on providing a framework for virtual material testing so that new kinds of experiments can be built or existing experiments can be extended in the future. We present our solution for the *VirLab* by applying software engineering approaches toward its development. In this thesis, we will demonstrate how software engineering approaches can be applied to building a component-based application for the *VirLab* system. Commonality analysis is one approach to identifying commonalities and variabilities for the system. The commonality analysis provides what we need for the design stage, so we do not go through the stage of software requirement gathering and analysis. Based on commonalities and variabilities summarized in the commonality analysis, we develop the system architecture design and propose a component-based design. From the software side, the thesis scope is from the commonality analysis stage, the design stage to the implementation stage. Each stage is documented and attached as an appendix. These appendices include a module guide, a module interface specification and a component description.

1.6 Thesis organization

Chapter 2 presents the commonality analysis for the real material experiments. It provides the terminology related to material experiments and also provides the

terminology for constitutive equations and then summarizes the commonalities and variabilities. Chapter 3 presents how software engineering approaches can be applied to building a component-based application for the *Virlab* system. This chapter also presents each type of document produced during the design phase. Chapter 4 provides an overview and examples of the actual *Virlab* system. Chapter 5 presents conclusions and contributions of this work and gives suggestions for future work. The documents that were produced during the design phases can be found in the appendices.

Chapter 2

Commonality analysis of material experiments

In Chapter 1, we provided an overview of material testing and decided that there are so many material tests that we needed to reduce the scope of the thesis. We narrowed our interests to displacement-controlled experiments and load-controlled experiments. In each experiment we are interested in applying the precise specification of the constitutive equation to the material, obtaining the kinematics quantities and stress, and plotting and analyzing the experimental data.

To develop a virtual laboratory for material testing, a close examination of real experiments is required. We make an examination by the commonality analysis of the displacement-controlled and load-controlled experiments. Commonality analysis is one approach to identifying commonalities (assumptions that are true for the current system and even future versions of this system), variabilites (assumptions about what can vary among different versions of the system) and common terminology for the system [AW97].

This chapter starts by providing the terminologies we use in the descriptions of the load-controlled and displacement-controlled experiments in Section 2.1. We

then analyze the commonalities between these two experiments in Section 2.2. In Section 2.3, we give a displacement-controlled experiment as a case study and then summarize the variabilities between the load-controlled experiments and the displacement-controlled experiments. Finally, we provide conclusions after the commonality analysis in Section 2.4.

2.1 Terminology

This section provides a set of technical terms used in the discussion about and description of the virtual laboratory for material testing.

Point/Particle

The term “point” is used exclusively to designate a location in a fixed space; that is, a point is a place in space. The term “particle” denotes a small volumetric element of a continuum; that is, a particle is a small part of a material continuum[Maz70].

Deformation

The term “deformation” refers to a change in the shape of the continuum between an initial (undeformed) configuration and a subsequent (deformed) configuration[Maz70].

Continuum configuration

The initial configuration, which is also called the reference configuration, is referred to as a Lagrangian coordinate system, written as $OX_1X_2X_3$. The final configuration, which is also called the deformed configuration, is referred to using an Eulerian coordinate system, written as $ox_1x_2x_3$ [Maz70]. We use a general example to explain the difference between the Lagrangian coordinate system and

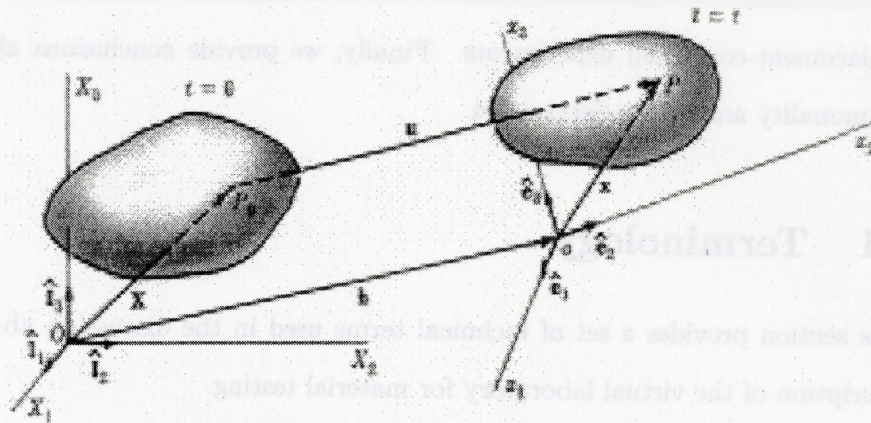


Figure 2.1: Coordinate system [Maz70]

the Eulerian coordinate system. In Figure 2.1 the Lagrangian coordinate system of a material continuum at time $t = 0$ is shown together with the Eulerian coordinate system of the same continuum at a later time $t = t$. For a fixed particle of the continuum, the Lagrangian coordinate is used to record the location at the undeformed configuration and the Eulerian coordinate is used to record the location at the deformed configuration. The purpose of two configurations is to describe the position of a particle in the original and deformed configuration and also to determine the relative change in the position of particles over time, so as to characterize the deformation of the material.

Displacement versus load controlled experiment

Displacement-controlled experiments and load-controlled experiments are categorized as the experiment class. The displacement-controlled experiment refers to

when the test specimen is extended or compressed by a displacement and the displacement works as the independent variable. Figure 2.2 schematically shows a displacement-controlled experiment. In the load-controlled experiment, the test specimen is extended or compressed by a load acting on the testing specimen and in this case the load works as the independent variable. Figure 2.3 schematically shows a load-controlled experiment. In Figures 2.2 and 2.3, a rectangle means the test specimen, a dotted rectangle represents the test specimen after the deformation caused by an applied load or displacement on the test specimen, x_1 , x_2 , X_1 and X_2 represent the axial directions, L_0 and W_0 represent the original length and width, respectively; L and W stand for the updated length and width respectively, u is the displacement at the end of the body in the axial direction, \dot{u} represents the velocity function of time and f represents the load function of time. The circle represents a roller and the triangle represents a pinned support. A pinned support means that the point on the test specimen is fixed from translation in either direction, although it could rotate if the other supports allowed this motion. A roller means that the point on the test specimen can translate only in one direction. For example, a roller in x direction means the point can only translate in the x direction.

Uniaxial, biaxial and multiaxial experiments

In each experiment class, from the view of the directions in which the test specimen is extended or compressed, experiments are further classified as the experiment type: uniaxial, biaxial or multiaxial experiments. In a uniaxial experiment, as shown in Figure 2.2 and Figure 2.3 the test specimen is extended or compressed in one axial direction. In the biaxial experiment shown in Figure 2.4, the test

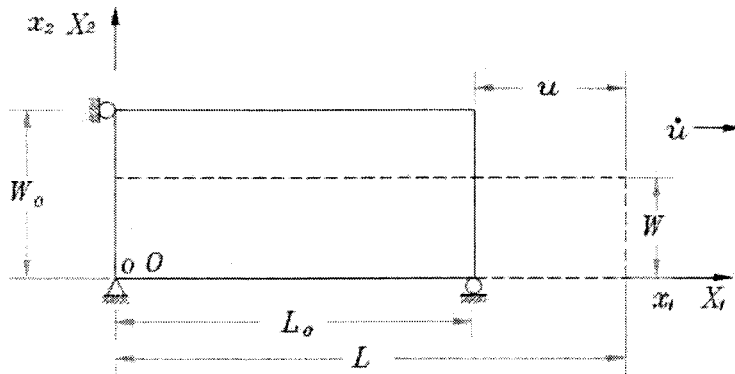


Figure 2.2: A displacement-controlled experiment

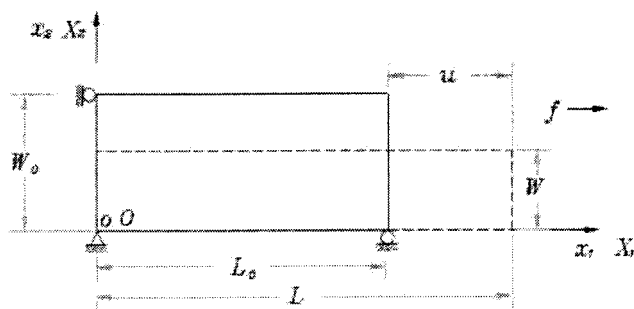


Figure 2.3: A load-controlled experiment

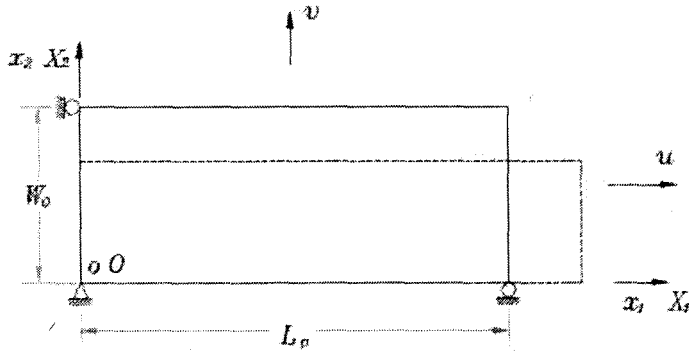


Figure 2.4: A biaxial experiment

specimen is extended or compressed in two axial directions and in the multiaxial experiment shown in Figure 2.5 the test specimen is extended or compressed in all three axial directions.

Kinematics quantities

Kinematics quantities deal with the deformation of the test specimen in response to the external factors such as the displacement or force acting on the test specimen. The following are classified as kinematics quantities: material deformation gradient, spatial deformation gradient, material displacement gradient, spatial displacement gradient, Cauchy's deformation tensor, Green's deformation tensor, Lagrangian infinitesimal strain tensor, Eulerian infinitesimal strain tensor, Lagrangian finite strain tensor, Eulerian finite strain tensor, true strain tensor, stretch tensor and stretch ratio tensor [Maz70][AJ96]. Appendix A lists how to calculate these kinematics quantities.

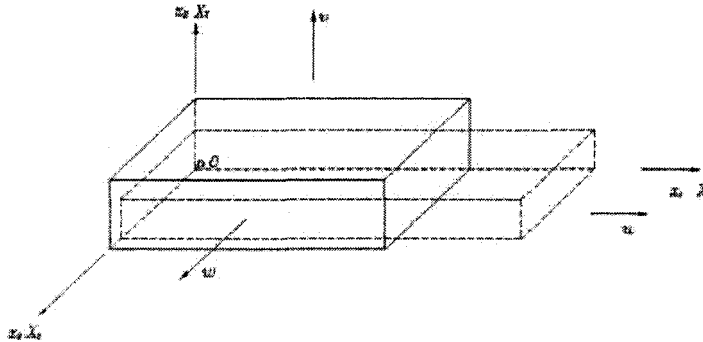


Figure 2.5: A multi-axial experiment

Strain

The strain is closely related with the deformation, as the strain provides a quantitative measure of the relative displacement between neighboring particles of the material. The strain is caused by the action of the displacement or force on the test specimen. There are many kinds of strains, such as shearing strain, normal strain, engineering strain, true strain, *etc.* We use Figure 2.2 to illustrate how to calculate the engineering strain and true strain. Appendix A summarizes how to calculate other strains.

Engineering strain

$$\epsilon = \frac{u}{L_0}$$

True strain [Smi93]

$$\epsilon_t = \Sigma \Delta \epsilon = \Sigma \left(\frac{\Delta L}{L} \right) = \int_{L_0}^L \frac{dL}{L} = \ln \left(\frac{L}{L_0} \right)$$

From the above definitions, the true strain is calculated by adding successive value

of $\Delta\epsilon$, which are obtained by dividing each increment ΔL of the distance between the gage marks by the corresponding value of L . Instead, the engineering strain is calculated by using the total elongation u and the original value L_0 [BJ81]. When the deformation is small, the difference between the true strain and the engineering strain is negligible, but when the deformation of the specimen is large, the true strain provides more information about the deformation history.

Stress

“Stress is the internal resistance in a body to the external forces or reactions acting on the body” [Pet69]. There are many kinds of stresses, such as shearing stress and bearing stress *etc.* We will focus on the engineering stress and the true stress in the virtual laboratory. Figure 2.6 shows the elongation of a rod subjected to a uniaxial tensile force F . Part (a) shows the rod with no force on it; Part (b) shows the rod subjected to a uniaxial tensile force F which elongates the rod from L_0 to L . The engineering stress σ on the bar is equal to the average uniaxial tensile force F on the bar divided by the original cross-sectional area A_0 of the bar.

Engineering stress is calculated via

$$\sigma = \frac{F}{A_0}$$

The true stress σ on the bar is equal to the average uniaxial tensile force F on the bar divided by the new cross-sectional area A after the elongation of the bar.

True stress σ_t [Smi93] is calculated by

$$\sigma_t = \frac{F}{A}$$

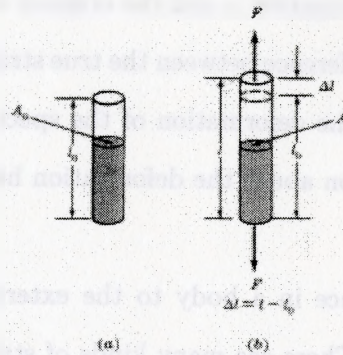


Figure 2.6: Elongation of a rod subjected to a uniaxial tensile force

The difference between the engineering stress $\sigma = \frac{F}{A_0}$ and the true stress $\sigma_t = \frac{F}{A}$ is that the instantaneous cross-sectional area A of the deformed specimen is used in the calculation of the true stress, so the true stress is more related with the deformation history. As we discussed above for the true strain and the engineering strain, the true stress is also related with the deformation history. Therefore, the true stress and the true strain together reflect more accurately the behaviour of the material.[BJ81]

Constitutive equation

A constitutive equation serves the role of an equation to solve the required kinematics quantities. Sometimes the conservation equation of physics are not enough to solve for the required unknowns; therefore, we need to add another equation that provides information relating the unknowns through characteristic material properties. Unlike a generally applicable equation, such as the conservation of

momentum equation, a constitutive equation is tied to the specific material that it is developed for.

A constitutive equation is a mathematical equation on internal attributes of the material and describes the macroscopic behaviours resulting from the internal constitution of the material and specially characterizes the individual material [Mal69]. We know that the range of materials is wide, for instance, elastic material, viscous material, viscoelastic material, plastic material, *etc.* When the entire range of possible temperatures and deformations is considered, materials behave in such complex ways that it is not feasible to write down one equation or set of equations to describe accurately a real material over its entire range of behavior. Instead, the constitutive equation for a material approximates physical observations of a real material's response over a suitably restricted range [Mal69].

Table 2.1 generally introduces the elastic, viscous, viscoelastic and plastic materials as possible constitutive models. The difference between elastic and viscous or plastic materials is that elastic material returns to an undeformed state upon removal of an applied force. Viscous or plastic materials, however, have no tendency for deformation recovery. For a viscous material the deformation occurs over time, even under a constant load. In the model of a plastic material the permanent deformation does not depend on time, but on the magnitude of the loading. For a plastic material it is necessary to track the loading history to determine how the material will respond to a new loading.

The examples of constitutive models are by no means exhaustive. In the current overview we have left out orthotropic materials, where the material response is different depending on the direction of loading. We have also left out materials

that have a deformation history dependent damage parameter that is used to capture how the material fatigues with repeated loading. A more detailed treatment of constitutive equation can be found elsewhere [Maz70].

Given the variety of models, it is hard to describe all materials in one constitutive equation. We could describe all materials using the principle of determinism for stress: “The stress in a body is determined by the history of the motion of that body [Mal69].” Unfortunately, this statement is too abstract to be of value for concrete implementation. We need then to adopt different models for different materials and the algorithms for applying the constitutive equations will be significantly different. Moreover, the algorithm for using the constitutive equation will change, even for the same material model, depending on whether we are conducting a displacement or a load controlled experiment.

2.2 Commonalities

“Identifying common aspects of the family is a central part of the analysis, accordingly, a commonality analysis contains a list of assumptions that are true for all family members. Such assumptions are called commonalities” [AW97]. Since a multiaxial experiment is the most complex experiment in our system, common assumptions are based on the multiaxial experiment. These assumptions are also used in the uniaxial experiment and the biaxial experiment.

Figure 2.7 schematically shows a multiaxial experiment. In this figure the rectangular box represents the test specimen used in the experiment. The dashed

Material name	Material description
Elastic material	If the strains caused in a material by the application of a given load disappear when the load is removed, this material is called an elastic material. The ideal linear elastic material is assumed to obey Hooke's law in which a uniaxial stress situation takes the form $\sigma = E\epsilon$ expressing a linear relation between the axial stress and strain, where E is the modulus of elasticity [Maz70].
Viscous material	A viscous material is assumed for many applications in fluid analysis. The constitutive equation for a fluid relates the rate of deformations D to the applied stress. It is generally assumed that the viscous stress is a function of the rate of deformation D , that is, $\tau = f(D)$. For a so-called Newtonian fluid, this relation is linear, $\tau = \eta D$, where τ is the stress, η is the viscosity and D is the rate of deformation [Maz70].
Viscoelastic material	A viscoelastic material is characterized by possessing both viscous and elastic behavior. The Maxwell equation $\sigma + \lambda \dot{\sigma} = 2\eta \dot{\epsilon}$ is one kind of constitutive equations for a viscoelastic material, where λ is called the relaxation time
Plastic material	If the strains caused in a material by the application of a given load do not return to zero after the load has been removed, this material is called plastic material. Four examples of idealized plastic behaviors include: rigid-perfectly plastic, in which elastic response and work-hardening are missing completely, elastic-perfect plastic, in which elastic response prior to yield is included but work-hardening is not, rigid-linear-work-hardening plastic, in which elastic response is omitted and the work-hardening is assumed to be linear, and elastic-linear-work-hardening plastic, in which elastic response prior to yield is included and the work-hardening is assumed to be linear [Mal69].

Table 2.1: A comparison of different material models

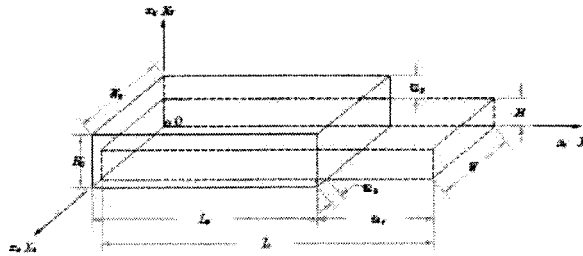


Figure 2.7: A multi-axial experiment for the commonality analysis

rectangular box represents the test specimen after the deformation. $ox_1x_2x_3$ represents the Eulerian coordinate system. $OX_1X_2X_3$ stands for the Lagrangian coordinate system. $\bar{u}_1, \bar{u}_2, \bar{u}_3$ refer to the displacements at the end of the body in the direction of the coordinate axes. L_0, H_0, W_0 represent the initial length, height and width of the test specimen and L, H, W represent the new length, height and width of the test specimen after the deformation, respectively. In this experiment the test specimen is extended or compressed by the external factors such as the displacement or force from three directions. Based on this experiment, commonalities are organized into the following list.

- The test specimen, used in the experiments, is assumed to be a rectangular box. Length, width and height of the rectangular box is changeable, but the shape of the rectangular box is fixed, that is, after the deformation the shape of the test specimen is still a rectangular box.
- Since in the experiment we are interested in kinematics quantities, stresses and the constitutive equation, we assume that material properties of the test

specimen, such as Young's modulus [BJ81] are known. That is, material properties can be specified; for instance, the properties can be specified through the user interfaces in the virtual laboratory.

- Lagrangian coordinate system ($OX_1X_2X_3$) and the Eulerian coordinate system ($ox_1x_2x_3$) are superimposed as shown in the Figure 2.7. That is, the origins of these two coordinate systems are the same and the coordinate axes are coincident.
- In the Lagrangian coordinate system and Eulerian coordinate system, there are some assumptions on relationships among the variables as follows:

$$- L = L_0 + \bar{u}_1, H = H_0 + \bar{u}_2, W = W_0 + \bar{u}_3$$

$$- x_1 = X_1 + u_1, x_2 = X_2 + u_2, x_3 = X_3 + u_3$$

The variables, u_1, u_2, u_3 , represents displacements along the coordinate axes in the period of the deformation and they are functions of either the Eulerian or Lagrangian coordinates. X_1, X_2, X_3 represents the location of a particle of the test specimen in the Lagrangian coordinate system. x_1, x_2, x_3 represents the location of a particle in the Eulerian coordinate system.

- u_1, u_2, u_3 are assumed as a function of X_1, X_2, X_3 .
 - The displacements u_1, u_2, u_3 are zero at $X_1 = 0, X_2 = 0$ and $X_3 = 0$, respectively
 - The displacements at the end points are $u_1 = \bar{u}_1, u_2 = \bar{u}_2, u_3 = \bar{u}_3$ at $X_1 = L_0, X_2 = H_0$ and $X_3 = W_0$, respectively

- The displacements are assumed to vary linearly through the test specimen $u_1 = \bar{u}_1 \frac{X_1}{L_0}, u_2 = \bar{u}_2 \frac{X_2}{H_0}, u_3 = \bar{u}_3 \frac{X_3}{W_0}$
 - $\vec{f} = (f_1, f_2, f_3)$ is a vector representing the forces applied to the sides of the test specimen
 - $\vec{u} = (\bar{u}_1, \bar{u}_2, \bar{u}_3)$ is a vector representing the displacement at the end of the specimen
- In the case of the load-controlled experiment the load is an independent variable and in the other case of the displacement-controlled experiment the displacement takes the role of the independent variable. Regardless of whether it is the load or the displacement, the independent variable is assumed to be a function of time (t). For example, the displacement function $\bar{u}(t)$ for a displacement-controlled experiment can be a function of time or it can be supplied as a velocity function of time $\dot{\bar{u}}(t)$ or as an acceleration function of time $\ddot{\bar{u}}(t)$. The load function is supplied as a force function of time $\vec{f}(t)$ or the first time derivative of force $\dot{\vec{f}}(t)$ or the second time derivative of force $\ddot{\vec{f}}(t)$.
 - In the virtual laboratory experiments, shear is not considered.
 - In the virtual laboratory experiments, only the displacement or the load is an independent variable. Experiments are not allowed to simultaneously set both of them for a given direction.

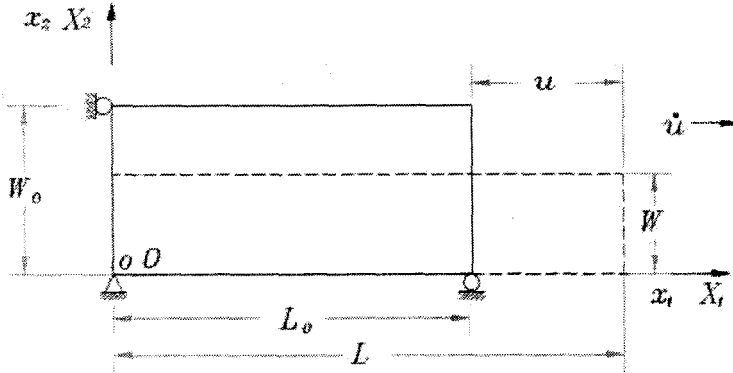


Figure 2.8: A uniaxial displacement-controlled experiment

2.3 A case study: a uniaxial displacement-controlled experiment for a viscoelastic material

We adopt a uniaxial displacement-controlled experiment as a case study to illustrate what we want to obtain from a real experiment. Figure 2.8 schematically shows a uniaxial displacement-controlled experiment. In this experiment the test specimen is assumed incompressible. That is, $LHW = L_0H_0W_0$ and the test specimen is extended by the displacement function which is given by $\dot{u} = \xi L_0 e^{\xi t}$, where ξ is a constant, the initial length (L_0) and width (W_0) of the test specimen are known. The function for \dot{u} was selected so that a constant strain rate will be obtained. For this example the Maxwell constitutive equation for a viscoelastic material is assumed.

	$t < 0$	$0 \leq t \leq MAXTIME$		$t > MAXTIME$
		$\text{inrange}(t, L_0)$	$-\text{inrange}(t, L_0)$	
\dot{u}	0	$\xi L_0 e^{\xi t}$	0	0

Table 2.2: The specification of the displacement function in the uniaxial experiment

	$t < 0$	$t \geq 0$
σ	0	$\sigma + \lambda \dot{\sigma} = 2\eta \dot{\epsilon}(t)$

Table 2.3: The specification of Maxwell's equation

To clearly specify the requirement of this uniaxial displacement-controlled experiment, we use two tables to illustrate the specifications of the displacement function and the Maxwell constitutive equation. Table 2.2 shows the specification of \dot{u} . In this table inrange is defined as $\text{inrange}(t, L_0) \equiv (-L_0 < u(t) < MAX_STRETCH_RATIO * L_0)$ where $MAX_STRETCH_RATIO$ is a constant representing the maximum stretch. In the definition of inrange , $u(t)$ can be calculated by

$$u(t) \equiv \int_0^t \dot{u}(\tilde{t}) d\tilde{t} \tag{2.3.1}$$

From the Table 2.2 the domain of the displacement function is specified clearly. The specification of the Maxwell constitutive equation is shown in the Table 2.3. In this equation, the stress and strain refers to the true stress and true strain. True strain can be calculated by

$$\epsilon(t) = \ln\left(\frac{L(t)}{L_0}\right) \tag{2.3.2}$$

Based on assumptions discussed in the last section, an additional equation can be

Geometry	Material Parameters	Time
$L_0 = 1(m)$	$\lambda = 0.01(s)$	$t_0 = 0(s)$
$W_0 = 0.1(m)$	$\eta = 2000(Pa.s)$	$t = 50(s)$
$H_0 = 0.1(s)$	$\xi = 0.8(m/s)$	$\Delta t = 0.01(s)$

Table 2.4: Assumed parameters for the Maxwell's equation

setup as follows:

$$L(t) = L_0 + u(t) \quad (2.3.3)$$

Using the assumed incompressibility of the material we obtain the following equation:

$$A(t)L(t) = A_0L_0 \quad (2.3.4)$$

Based on the constitutive equation Table 2.3 and equations 2.3.1, 2.3.2, 2.3.3 and 2.3.4 and the parameters for the Maxwell equation shown in Table 2.4, u , L , W , ε and σ can be calculated. Appendix B provides the detailed steps. Once u is known, based on the relationships between u and the kinematics quantities shown in the table in Appendix A, kinematics quantities can be obtained. However, if we mistakenly use the engineering strain and engineering stress to solve the Maxwell constitutive equation, the results are totally different and wrong. Figure 2.9 shows the difference between the two approaches. Two errors occurred in the incorrect process the engineering definitions were mistakenly used for σ and ε and then the mistake was compounded by converting the engineering values to true values. In Figure 2.9 the dotted line represents the relationship between the true strain and true stress that are used to solve the Maxwell constitutive equation. The solid line represents the relationship between the true strain and true stress that were

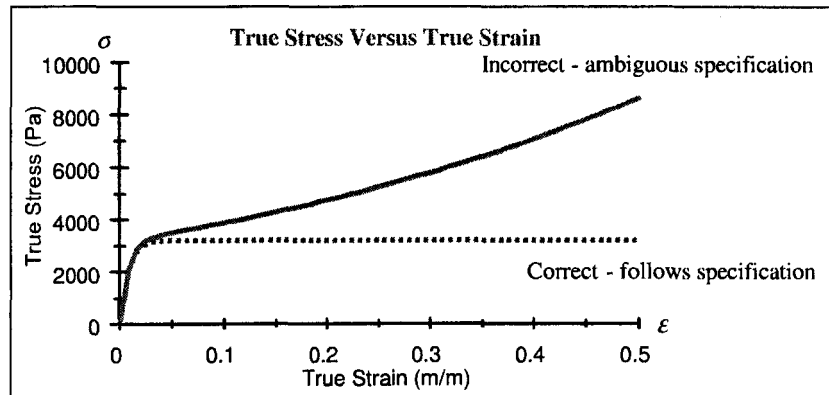


Figure 2.9: Comparison of data for the correct and incorrect experiment

found using the incorrect approach. Figure 2.9 illustrates that the correct specification of the experiment is important; therefore, it is worthwhile to invest in a virtual environment to help researchers and students understand the importance of specification and of using the correct stress and strain measures. The virtual laboratory should prevent user from making mistakes of the sort illustrated here.

From the above example, we can draw the following conclusions:

- The displacement function can be determined from a first derivative function of time (t), that can also be understood as a velocity function of time (t), $\dot{u}(t)$.
- During the approximation of the constitutive equation, the Maxwell constitutive equation in this case, a specialized algorithm must be adopted.
- The initial geometric information of the test specimen, such as the original length (L_0) and width (W_0), can be assumed to be given.

- The material properties, such as η and λ , are given.

In the case study a displacement-controlled experiment is shown. There are two obvious differences between the displacement-controlled experiment and the load-controlled experiment. One is the independent variable. The other is how to solve the constitutive equation. In the displacement-controlled experiment, since the displacement is given, the kinematics quantities history can be calculated directly from the displacement. Since the constitutive equation is an equation that describe the relationship between the stress and the kinematics quantities, the stress can be approximated after the kinematics quantities history is obtained. However, in the load-controlled experiment, the load is known so it is the stress, not the kinematics quantities, that are approximated. Moreover, only the engineering stress can be immediately determined, as the true stress requires knowledge of the deformed configuration, but the displacement and the kinematics quantities are still unknown. The constitutive equation in one direction is not enough to solve for all the unknowns, other equations are required to setup.

The complication of the load-controlled experiment can be illustrated by an example that assumes that Hooke's law $\sigma_x = E\varepsilon_x$ is given. In the uniaxial displacement-controlled experiment, since kinematics quantities can be calculated from the known displacement, the strain is easily calculated. Based on $\sigma_x = E\varepsilon_x$, the stress can also be calculated in a straightforward manner. However, in the uniaxial load-controlled experiment, only the stress (engineering stress) can be calculated from the known load. If the constitutive equation $\sigma = E\varepsilon$ is for true values, then the strain can not be calculated directly. To calculate the strain and other kinematics quantities, other equations are required. In this case we could use Poisson's ratio

ν to characterize how the material deforms in the other coordinate direction so that $\varepsilon_y = -\nu \frac{\sigma_x}{E}$ and $\varepsilon_z = -\nu \frac{\sigma_x}{E}$. Using this information, we can solve for the true stress and for the kinematics quantities.

Although the Hooke's law is a simple constitutive equation for the small strain linearly elastic material, the procedure to solve the constitutive equation in the displacement-controlled experiment and the load-controlled experiment is different. Therefore, procedures to solve the complex constitutive equations in the load-controlled experiment and the displacement-controlled experiment are significantly different. This fact will need to be accounted for in the design presented in Chapter 3.

Although there are difference in how the constitutive model is introduced, there are many similarities between the displacement-controlled experiment and the load-controlled experiment. For instance, kinematics quantities, stresses and the analysis of the result data are handled the same for both classes of experiments. Therefore, we extend our thoughts to all the experiments as follow:

- Regardless of the displacement or the load in the experiment, the similarity is that they are from some kind of functions of time (t), *e.g.* a velocity function of time $\dot{u}(t)$, an acceleration function of time $\ddot{u}(t)$, a force function of time $\vec{f}(t)$, the first derivative of force $\dot{\vec{u}}(t)$ or the second time derivative of force $\ddot{\vec{u}}(t)$. It is beyond the scope of this thesis to determine an abstract form to represent all possible mathematical functions. So we will stay with three popular types of functions: a quadratic function, a sin/cosine function and an exponential function.
- It would be best to find an abstraction that would allow us to describe

all constitutive equations. However, we decided not to do this because an abstraction for all constitutive equation is challenging. Moreover, people should have some solid background on materials test to undertake this task. Therefore, the task for the approximation of the constitutive equation will be delegated to professionals, as will be shown in Section 3.6.2.

Based on the above discussion, we predict the possible changes over the lifetime of the program from the view of variabilities. They are organized into a list where each item identifies a separate concern:

- The shape of the test specimen before and after the deformation has been assumed as a rectangular box, but the way to specify the geometric information of the test specimen such as the original length, width and height are changeable.
- The material properties are assumed to be given, but the way to specify the material properties are changeable. For example, material properties are given in the way of a file or input directly from the external devices such as mouse or keyboard.
- Since procedures to approximate the constitutive equation in the displacement-controlled experiment and the load-controlled experiment are different, the designer shown in Chapter 3 will use one module for the approximation of the constitutive equation in the displacement-controlled experiment, and another is for the approximation of the constitutive equation for the load-controlled experiment. In each module algorithms used to approximate the constitutive equation are changeable.

- Algorithms to calculate or approximate the kinematics quantities and stress are changeable.
- The ways to output the result data are changeable. For example, the result data could be output as curves or as tables.

2.4 Conclusions

To develop a virtual laboratory for material testing, which will be named as *Virllab*, we have analyzed the real material experiment using a commonality analysis approach. In addition, we have summarized the common terminologies for communicating and identifying the commonalities, or unlikely changes, shown in the Table 2.6 and the variabilities, or anticipated changes, shown in the Table 2.7. After the commonality analysis, the next step is typically a requirements analysis, followed by the creation of a requirements document. However, since *Virllab* is a comparatively small system, a requirement document was not explicitly created. Rather than a full requirements document, we will provide a brief overview of how the system can be imagined. To describe the system, we will use Table 2.5, which is assumed to represent the output of experimental result data. From the output view of the data, the u column will be filled in the displacement-controlled experiment and the f column will be filled in the load-controlled experiment. After the experiment, the other columns will be filled. For example, after the experiment the kinematics quantities represented with ε_t will be calculated and filled into the table. To fill in the table, we need to specify the following information.

- We need to specify the required experiment information corresponding to

t	u	f	ε_t	ε_e	σ_t	σ_e	...
t_0	<i>undefined</i>
$t + \Delta t$
$t + 2\Delta t$
...
...

Table 2.5: Visualizing the *Virlab* system

label C.11 in the Table 2.7, *e.g.*, the specification of experiment type.

- We need to design data structures to represent those required information corresponding to labels C.1, C.2, C.6, C.7, C.8, C.9, C.10, C.12 and C.13 in the Table 2.7.
- We need to design corresponding algorithms to calculate displacement, kinematics quantities and stress, and to analyze the experimental data corresponding to labels C.3, C.4, C.5 in the Table 2.7.
- We need to consider the output of data corresponding to the label C.13 in the Table 2.7.

Although a requirement document was not explicitly created, we did consider the requirements before designing *Virlab*. We considered the requirements from two views: functional requirements and non functional requirements. For example, for functional requirements, we knew that *Virlab* will provide some functions, *e.g.*, calculating kinematics quantities, calculating stresses. For non-functional requirements, we consider that *Virlab* should be easily maintained. We keep system requirements in mind during the whole procedure instead of writing them

down as an independent documentation. With the determined commonalities and variabilities and our understanding on requirements, we can now go to the next stage - design of the *Virlab*.

Order	Unlikely changes (Assumptions)	Label
1	The shape of the test specimen before and after the deformation is a rectangular box	SHAPE
2	The displacement or load function is chosen from three popular functions: a quadratic function, a sin/cosine function or an exponential function	FUNCTION
3	The Lagrangian coordinate system and Eulerian coordinate system are superimposed.	SUPERIMPOSED
4	In the experiments the shear is not considered.	SHEAR
5	<ol style="list-style-type: none"> 1. The signs for Lagrangian coordinate system $OX_1X_2X_3$ and Eulerian coordinate system $ox_1x_2x_3$ are unchangeable. 2. $x_1 = X_1 + u_1, x_2 = X_2 + u_2, x_3 = X_3 + u_3$ 3. $L = L_0 + \bar{u}_1, H = H_0 + \bar{u}_2, W = W_0 + \bar{u}_3$ 4. $\bar{u}_1, \bar{u}_2, \bar{u}_3$ are assumed as a function of X_1, X_2, X_3. The displacements u_1, u_2, u_3 are zero at $X_1 = 0, X_2 = 0$ and $X_3 = 0$, respectively. 5. The displacements $u_1 = \bar{u}_1, u_2 = \bar{u}_2$ and $u_3 = \bar{u}_3$ at $X_1 = L_0, X_2 = H_0$ and $X_3 = W_0$ 	MATH_RELATION

Continued on next page

Continued from previous page

Order	Unlikely changes (Assumptions)	Label
	6. The displacements are assumed to vary linearly $u_1 = \bar{u}_1 \frac{X_1}{L_0}, u_2 = \bar{u}_2 \frac{X_2}{H_0}, u_3 = \bar{u}_3 \frac{X_3}{W_0}$	
6	The development environment or operating system supports mouse and keyboard function. That is, drivers for the mouse and/or keyboard exist.	DRIVER
7	The development environment or operating system provides the screen display functions such as buttons, combo lists and so on.	DISPLAY

Table 2.6: The list of unlikely changes

Order	Anticipated changes	Label
1	The geometric information of the test specimen is changeable.	C_1
2	The ways to specify the material properties are changeable.	C_2
3	Algorithms to approximate the constitutive equation are changeable	C_3
	Algorithms to approximate the constitutive equation in the displacement-controlled experiment are changeable	C_3_DISP

Continued on next page

Continued from previous page

Order	Anticipated changes	Label
	Algorithms to approximate the constitutive equation in the load-controlled experiment are changeable	C_3_LOAD
4	Algorithms to calculate the kinematics quantities are changeable:	C_4
	Procedure to calculate the material deformation gradient	C_4_MDG
	Procedure to calculate the spatial deformation gradient	C_4_SDG
	Procedure to calculate the material displacement gradient	C_4_MDPG
	Procedure to calculate the spatial displacement gradient	C_4_SDPG
	Procedure to calculate the Cauchy deformation gradient	C_4_CDG
	Procedure to calculate the Green deformation gradient	C_4_GDG
	Procedure to calculate the stretch tensor	C_4_ST
	Procedure to calculate the stretch ratio tensor	C_4_SRT
	Procedure to calculate the Eulerian infinitesimal strain tensor	C_4_EIST
	Procedure to calculate the Lagrangian infinitesimal strain tensor	C_4_LIST
	Procedure to calculate the Lagrangian finite strain tensor	C_4_LFST

Continued on next page

Continued from previous page

Order	Anticipated changes	Label
	Procedure to calculate the Eulerian finite strain tensor	C.4.EFST
	Procedure to calculate the true strain tensor	C.4.TST
5	Algorithms to approximate the stress	C.5
	Procedure to calculate the engineering stress tensor	C.5.ESST
	Procedure to calculate the true stress tensor	C.5.TSST
6	The way to present the constitutive equation for the purpose of display	C.6
7	The way to define the experiment	C.7
8	Although only one of three popular functions can be chosen as the displacement or load function, the coefficients of any of these functions are changeable	C.8
9	The way to describe the displacement	C.9
10	The way to describe the load	C.10
11	The ways to specify the required information for the experiment are changeable	C.11
	The way to specify the experiment definition	C.11.DEF
	The way to specify the displacement	C.11.DISP
	The way to specify the load	C.11.LOAD
	The way to specify the function's information	C.11.FUNC

Continued on next page

Continued from previous page

Order	Anticipated changes	Label
	The way to specify the geometric information of the test specimen	C.11_GEO
	The way to specify the constitutive equation	C.11_CON_EQU
12	The way to store the result data	C.12
13	The way to output the result data	C.13
14	The sequence of executing the program	C.14

Table 2.7: The list of anticipated changes

Chapter 3

Component-based design

In this chapter, we give the component-based design of the *Virlab* software system. The meaning of the term component is confusing without a clear definition, so Section 3.1 gives the definition of component that we use in the *Virlab* design. Section 3.2 discusses the reasons that the component-based idea comes up in the *Virlab* design. Since modularity is a prerequisite for components, Section 3.3 gives an overview of the modular design for the *Virlab* system and explains how to divide the system into modules. This section also gives the module hierarchy and use relation for the *Virlab*. Section 3.4 describes how to identify the components on the basis of modularity. Section 3.5 discusses the component design for the *Virlab* system based on the modular design for the system. Finally, Section 3.6 gives the documentation for the component description, the module guide and the module interface specification for the *Virlab* software.

3.1 What is a component?

Originally the word ‘component’ was used in the engineering field and referred to a constituent element, forming or functioning as part of a whole. Later this word

was borrowed by the software engineering field, but its meaning is often unclear because a concrete definition is not always supplied. However, a variety of definitions are available in the literature. Booch from the view of the source-level states: “A reusable software component is a logically cohesive, loosely coupled module that denotes a single abstract [Boo87]”. Jacobson states: “By components we mean already implemented units that we use to enhance the programming language constructs. These are used during programming and correspond to components in the building industry” [Jac93]. Jacobson’s component concept is general and macros or templates are thought of components. Sametinger states: “Reusable software component are self-contained, clearly identifiable pieces that describes and/or perform specific functions, have clear interfaces, appropriate documentation and a defined reuse status” [Sam97]. It is unfair and incorrect to judge whether their definitions are right and precise or not without knowing the context-sensitive environment where the definitions are used. The important decision in the current context is to adopt a definition for components and to use this definition consistently.

In this thesis a component is defined as a unit of composition that can be linked dynamically into the system. Along with explicitly specified interfaces, its composition includes other components and/or a set of modules that carry out a unique set of functional behavior. The definition of the component emphasizes its dynamical linking characteristic. The module is defined as a work assignment and its definition does not highlight dynamical linking characteristic. Since the component is a unit of composition, the relationship between the component and

the module is that a component might include single or multiple modules. Furthermore, components can be independently developed. This definition has the following characteristics:

- It embodies some part of the functionality of the system.
- Since a component can be independently developed and also has an explicit specified interfaces and binary executable form, it is a unit of distribution and configuration [KN96]. A unit of distribution refers to the fact that a component can be delivered independently and a unit of configuration means that a component can be used to configure the application.
- Because a component's composition can finish a subset of the functionality and it may include other components, it can be a subsystem [KN96].
- It is self-contained, which means the component itself communicates with outside components or applications by its interfaces and outside components and applications have no knowledge about its implementations; that is, the component's implementation is independent of other components.
- Explicit specified interface means that it needs to come with clear specifications of what it requires and what it provides. A component encapsulates its implementation and interacts with other components through well-defined interfaces [KN96].

The benefit of giving a component definition is that it is necessary to understand the meaning of the term component when a component is being used or described. Now that the component definition is clear, we can move to why the component-based idea is used in the *Virlab* software design.

3.2 Why use component-based system design?

Building a large software system is always difficult, so many researchers and practitioners devote themselves to research on how to reduce software design complexity and improve the correctness of systems. A component-based design is often used during the software development, especially for large systems, because the component-based design offers several advantages:

1. Possible reusability

During the development of the system, an outside component can be reused in different applications or other components. If an off-the-shelf component from a commercial organization, or existing projects, satisfies the part of the functionality of specified systems, it is better to adopt the known component into the system rather than design, code, test, debug and document a new component. Even in the same project, a good component can be reused in the different versions of the project. The obvious advantage of reuse is to reduce the development time and cut the costs of the project.

2. Increased reliability [Cle95]

An off-the-shelf component will be used in many other systems or different versions; therefore, it will be tested many times and many bugs will be found and sifted out over time. Reusing that component increases the reliability of the system.

3. Increased flexibility [Cle95]

Designing a system to accommodate the existing components means that the system has been built to ignore the details of the implementation of

those components. In such system any component that satisfies the system requirement with the same interface can easily replace the existing component.

4. Easy maintenance

If a system is built using components and each component carries out a unique subset of the system's functional behavior, independently of the other components, then when some modifications are made in the system, they will be localized. It is much easier to modify a few components than it is to change the entire system.

So far the advantages of component-based design have been discussed. It is natural to ask why component based design is suitable for the *Virlab* software design? The reasons will be explained based on the design decisions made for the *Virlab* software.

1. Rapid development

One design decision is that the system should be easily extended and contracted. The component-based design supports rapid development. In a component-based design, a new version of the system may reuse existing components or remove existing components without requiring modification of the other components. For example, assuming one version of *Virlab* exists that has both the load-controlled experiments and the displacement-controlled experiments implemented, a new version, just for special users to do displacement-controlled experiments, can be rapidly released by removing the load-controlled component.

2. Reusable components

Since the calculation of the kinematics quantities are needed not only in the displacement-controlled experiment but also in the load-controlled experiment, the decision was made that the calculation of the kinematics quantities and stress should be separated as independent components. That means that once the kinematics component and stress component are finished, they can be reused many times and should thus increase our confidence that errors have been found. Undoubtedly this will increase the reliability of system in later versions.

3. Extensive research

One purpose of the *Virlab* software is for the designer to do the research. (Please see the Section 3.6.2 for the definition of the designer.) If a designer wants to use a new constitutive equation to do the displacement-controlled experiment, he/she just needs to write a new component with the same interfaces described in the module interface specification of *displacement constitutive calculation module*. This new module will replace the existing displacement-controlled component, while the rest of the components in the system stay the same. So the component-based structure will make research on new constitutive models easier.

4. Easy update

From the view of a practical application, the *Virlab* software should be easy to maintain. Based on this consideration, the *Virlab* system is divided into several components based on the functionalities of the system. Each component might embody one functionality or part of the functionality.

Therefore, when the system needs to change, based on the function of the possible changes, the corresponding component may be quickly located and then modified. Because each component is relatively unique subset of the system's behavior, the modification might be limited to this component. So in the component-based system updates are easier than for a non-component based system.

5. Dynamic linking

One goal in the *Virlab* software is to dynamically plug components into and out of the *Virlab* system. It is important to support research without recompiling or relinking the updated or new component. For example, if a designer writes a new component for an experiment with a new constitutive equation, the end user will not be required to recompile and relink that component. The user will be able to add the component while the application is still running.

Since the component itself has so many advantages, such as increased reliability, reusability and flexibility, and our design decisions for the *Virlab* software provide an opportunity to build on these advantages, a conclusion can be made that the idea of the component-based design is suitable for the *Virlab* software.

So far the definition and benefits of components have been described, but the question remains how to partition our design into components? Partitioning a design into component is a careful process that has a significant influence on the success of the resulting components. From the composition of the component, modules can be thought of as the minimal components. So modularity is a prerequisite for the component. "Component technology unavoidably leads to modular

solutions” [Szy99].

3.3 Modularity

Since the components are rather close to modules, it is worthwhile to closely look into modules. A module is “a work assignment” [Par72] and “Modules are relatively self-contained systems that can be combined to make large systems” [HW01]. Modules are interconnectable and interchangeable parts. A modular architecture makes dependencies among the modules explicit and also should show a hierarchical structure. From such a layered architecture a natural distribution of responsibility becomes obvious. After modularity is built, it is easier to compose the components by following the principle of separation of concerns.

How to divide a system into modules is a challenge. Without careful consideration and iterative effort, a good modularity will not happen. Our goal was to decompose the system into a hierarchical structure.

3.3.1 System decomposition

System decomposition refers to the structure of the system architecture. “The primary goal of the decomposition into modules is reducing overall software cost by allowing modules to be designed and revised independently [PCW85]”. Information hiding [Par72] and separation of concerns are the principles we used for decomposing the system. In the decomposition a module is defined as “a work assignment”. Using the information hiding principle each module will contain some access routines and these routines will hide a design decision known as a

secret of the module from the view of the caller outside the module. Potential changes, such as the algorithms or data structures, are typical secrets that are hidden inside modules. By hiding the likely changes, it is possible to implement the likely changes in the futures, with a minimal amount of effort.

Figure 3.1 shows the system decomposition for the *Virlab*. In this figure, a rectangle with a double frame means this module has some sub modules, modules shown as a rectangle with a single frame are leaf modules, which are the modules that will be implemented, modules represented with a dotted frame means that they are provided by the operating system and the term mod is an abbreviation of the word ‘module’. To decompose the system into modules, we focus on several important aspects of the system. These aspects include sequence control, specification, data representation, independent functions and separate algorithms and data structures. They will be described in more detail below.

1. Sequence control

The *Virlab* version might vary in its system capacity. For example, will the *Virlab* provide the function of importing the new constitutive equation? Will the *Virlab* be used to do the displacement-controlled experiment and/or load-controlled experiment? Will the experiment in the *Virlab* be used to calculate the kinematics quantities and/or stress? From these questions, we therefore centralize the control of the experiment and assign this responsibility to the “*Experiment module*”. This module works as a mediator among *stress module*, *kinematics module*, *disp_con calculation module* and *load_con calculation module*. We also centralize the control of the flow of the system execution by assigning this responsibility to the “*Master control*”

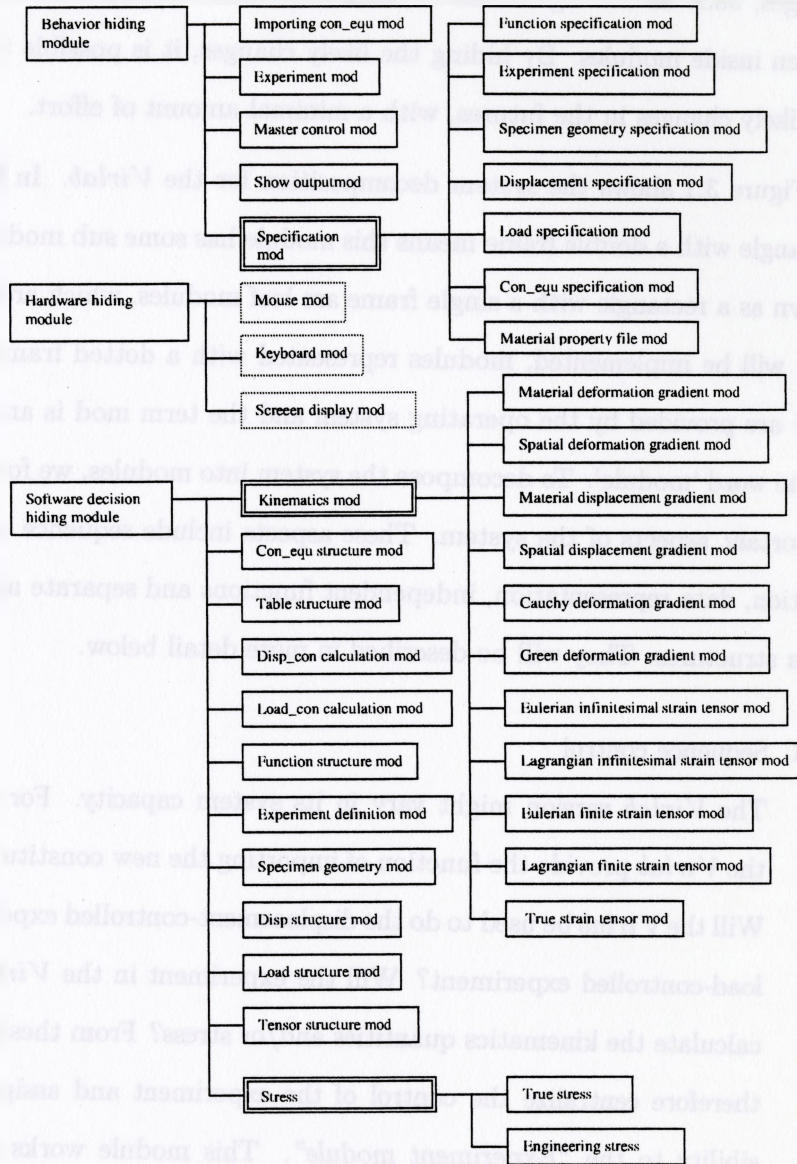


Figure 3.1: Module hierarchy

module". The *master control module* acts as a hub between the functions of the systems such as importing a new constitutive equation, doing the experiment and showing the output results of the experiment.

2. Specification

In the *Virlab*, a user needs to specify the information required by the experiment. The specification can be categorized as follows:

- Experiment definition information (*e.g.* experiment class (load versus displacement controlled experiment), experiment type (uniaxial, biaxial or multiaxial))
- Specimen information (*e.g.* the length, width and height of the specimen geometry)
- Function information (*e.g.* a function refers to a displacement function in the displacement-controlled experiment or a load function in the load-controlled experiment). The *Virlab* provides three popular functions: sine/cosine function, quadratic function and exponential function. The user should specify which type of functions is used in the experiment and corresponding parameters of each function used in the experiment.
- Displacement specification in the displacement-controlled experiment (*e.g.* which type of functions is chosen as the displacement function?)
- Load specification in the load-controlled experiment (*e.g.* which type of functions is chosen as the load function?)
- Constitutive equation information (*e.g.* select the constitutive model

from the list of available models)

- Material properties information (*e.g.* how many material properties are used in the constitutive equation and the numerical value of each material property)

How to specify the required information is related to the user interface of *Virlab* and also decides the friendliness of *Virlab* user interfaces. As a result, we delegate the responsibilities of specifying each type of information to each individual module that handles the *Virlab* user interface.

Once the required information is specified, it will be available for later use by the system. How to represent the above information for the further use is an important consideration.

3. Data representation

Putting the closely related data together is obviously a good idea, which makes the decomposition much cleaner. Therefore, a corresponding data structure for each type of specified information is designed. *e.g.* *specimen geometry module* deals with a data structure for the test specimen, *experiment definition module* presents a data structure to describe the experimental setup information, *displacement structure module* provides a data structure to represent the displacement, *load structure module* gives a data structure for the load on the specimen in the load-controlled experiment, *con_equ structure module* presents a data structure to describe the constitutive equation for the display and the *function structure module* gives a data structure to represent the functions. The *tensor structure module*

provides a data type for the tensor, without providing access routines. Because *kinematics module*, *stress module*, *disp_con calculation module* and *load_con calculation module* share a data type for the tensor, we believe it is better to separate it as an independent module.

4. Independent functions

Since designing the *Virlab* to be extensible and contractible is one of our goals and the *Virlab* version might vary in its system capabilities, predicting and designing the possible functions will lead to a good design. We consider that calculating the kinematics quantities and stress and approximating the constitutive equation should be totally or partly included in the system depending on the user's requirements. Since procedures to solve the constitutive equation in the displacement-controlled experiment and the load-controlled experiment are different as discussed in Section 2.3, we decide to handle these two cases separately. Hence, we predict four possible functions and delegate them into different modules: *kinematics module*, *stress module*, *disp_con calculation module* and *load_con calculation module*. The *kinematics module* is used to obtain the kinematics quantities. The secret of the *stress module* is now to calculate the stress. The *disp_con calculation module* serves to approximate the constitutive equation in the displacement-controlled experiment and the *load_con calculation module* is to approximate the constitutive equation in the load-controlled experiment.

5. Separate the algorithms and data structures

It is not always desirable to separate the algorithm and data structure. In

some domains it is hard to separate the algorithm from the data structure and in some domains it is also unwise to separate them because that means the system might lose efficiency and performance. However, one of the characteristics of the information hiding principle is that one module only has one secret. Because our intention is to apply information hiding principle to practical application and our system does not focus on improving system performance, we put considerable thought into how to handle the relationship between the algorithm and the data structure. We decided that the data structure is used to present the required data in the experiment. The module interfaces are carefully designed to output the required information by the access programs of the module. For example, the secret of the displacement constitutive calculation is an algorithm to approximate the constitutive equation in the displacement-controlled experiment. This module will use the specimen geometry information and the displacement information. The specimen geometry information will be provided by the access programs of the *specimen geometry structure module*, whose secret is a data structure to describe the specimen geometry. Displacement information will be provided by the access programs of the *displacement structure module*, whose secret is a data structure to represent the displacement.

3.3.2 Hierarchical structure

In addition to carefully designing the *Virlab* system decomposition through secrets, we also used a hierarchical structure for the design of *Virlab*. Simply, “we

have a hierarchical structure if a certain relation may be defined between the modules or programs and that relation is a partial ordering" [Par72]. The relation in the definition of hierarchical structure is referred to as a "use" relation. Program A uses program B means that "correct execution of B may be necessary for A to complete the task described in its specification. That is, A uses B if there exist situations in which the corrected [sic] functioning of A depends upon the availability of a correct implementation of B [PS75]". A hierarchical structure is defined as follows: "A relation or predicate on pairs of the parts ($R(\alpha, \beta)$) allows us to define levels by saying that

1. Level 0 is the set of parts α such that there does not exist a β such that $R(\alpha, \beta)$, and
2. Level i is the set of parts α such that
 - a. There exists a β on level $i - 1$ such that $R(\alpha, \beta)$ and
 - b. If $R(\alpha, \gamma)$ then γ is on level $i - 1$ or lower. [Par74]"

Figure 3.4 gives the modules that constitute the *Virlab* system and their uses relation [Par74]. A rectangle with a single frame means a module and a rectangle with a double frame means a module that has some sub modules and all the sub modules have the same use relation externally in the use relation of the system. A rectangle with a dotted frame means that a module is assumed to be implemented and provided by the operating system. *Kinematics Module* has some sub modules shown in the Figure 3.2. A significant characteristic of the use relation of the *kinematics module* is that all submodules in the *kinematics module* can use each

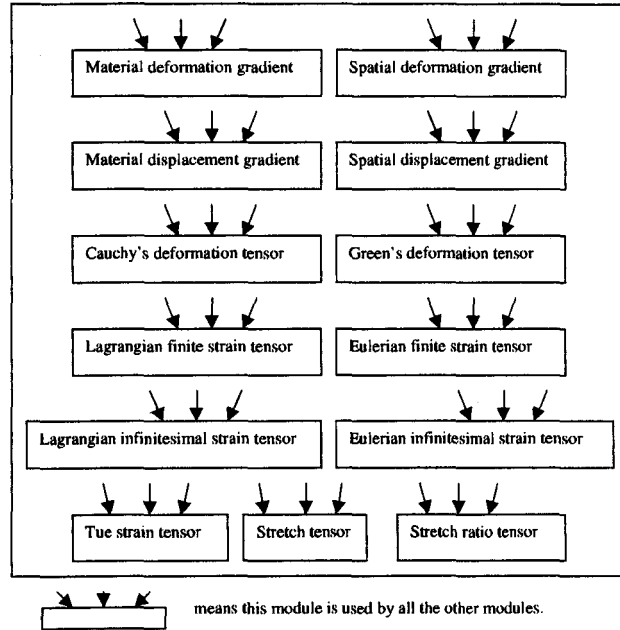


Figure 3.2: Use relation in the *kinematics module*

other because sometimes known kinematic quantities will be used to calculate an unknown kinematic quantity. All the submodules in the *kinematics module* have the same use relation externally with the other modules in the system. Following a similar argument, the use relation of the *stress module* shown in Figure 3.3 has the same characteristics as the use relation of the *kinematics module*; *i.e.*, all submodules in the *stress module* can use each other when a known stress is used to calculate an unknown stress and all the submodules in the *stress module* have the same use relation externally with the other modules in the system.

The *Virlab* system is constructed with a hierarchical structure by the use of the concept of transparency. The transparency in the system refers to a kind of

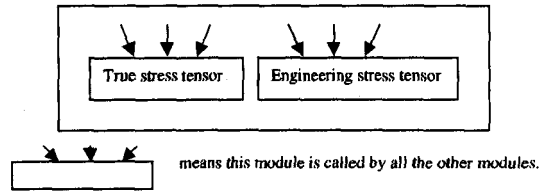
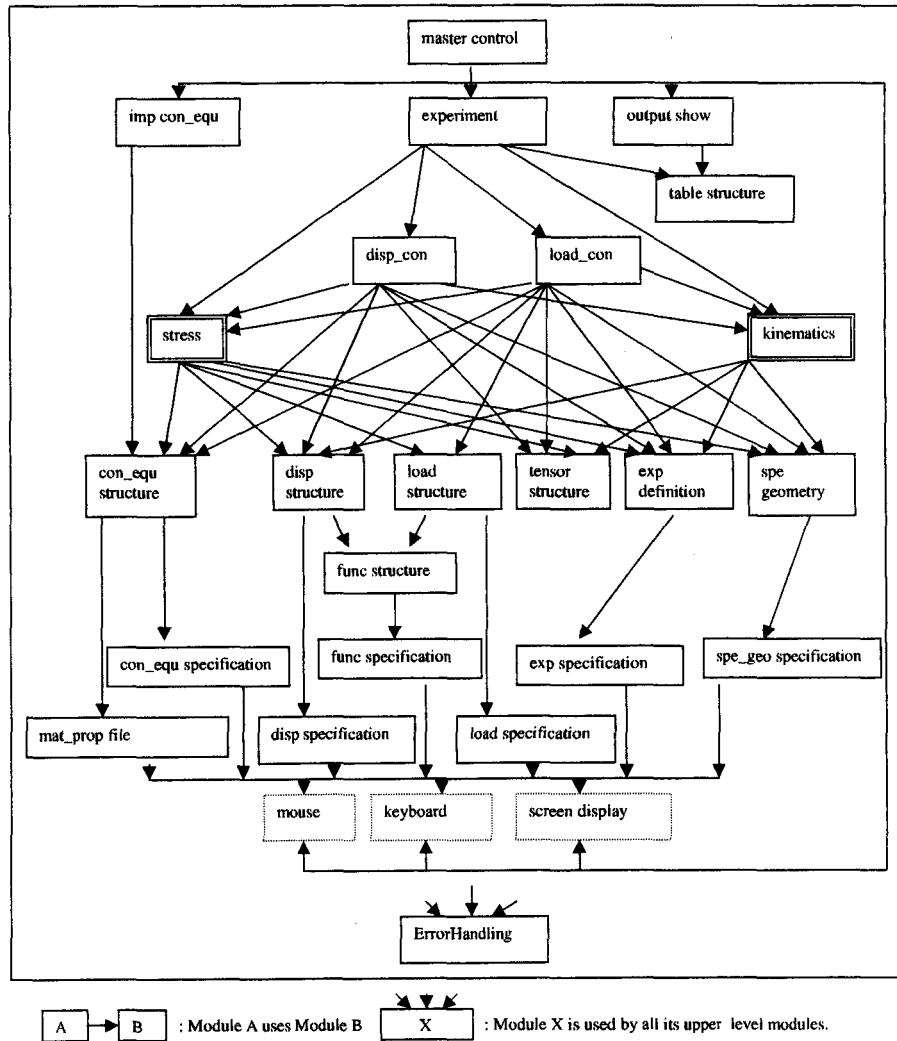


Figure 3.3: Use relation in the *stress module*

abstraction at each level in the system hierarchy. Each level in the hierarchical system provides a virtual machine which hides (or abstracts from) some aspects of the machine below it [PS75]. In the procedure of building the hierarchical structure we combine two approaches together: “Outside in [PS75]” and “Bottom up [PS75]”. The “Outside in” approach is adopted in the consideration of the functional behaviors. From the outside view the system should implement the functions such as calculating the stress and kinematics quantities and approximating the constitutive equation and then from the inside view we consider how to design each of them. The “Bottom up” approach is applied when the direction of data flow is considered. Figure 3.5 shows the data flow chart for an experiment. The required information is first input with a virtual device, such as the keyboard, by the ‘specification’ part to the corresponding data structures that are used to represent the required information. Secondly the ‘algorithm’ part uses the information from the corresponding data structure to do the calculation or approximation and result data is obtained. Thirdly the result data flow into a table. Finally the data result will be output to the concrete application. Compared with the direction of the use relation of the system shown in Figure 3.4, the direction of the data flow of the system shown in Figure 3.5 is from the bottom to the top.



Comment:
 disp_con, load_con, con_equ, mat_prop, spe_geo, func and exp are the abbreviations of displacement constitutive, load constitutive, constitutive equation, material property, specimen geometry, function and experiment.

Figure 3.4: Use relation of the system

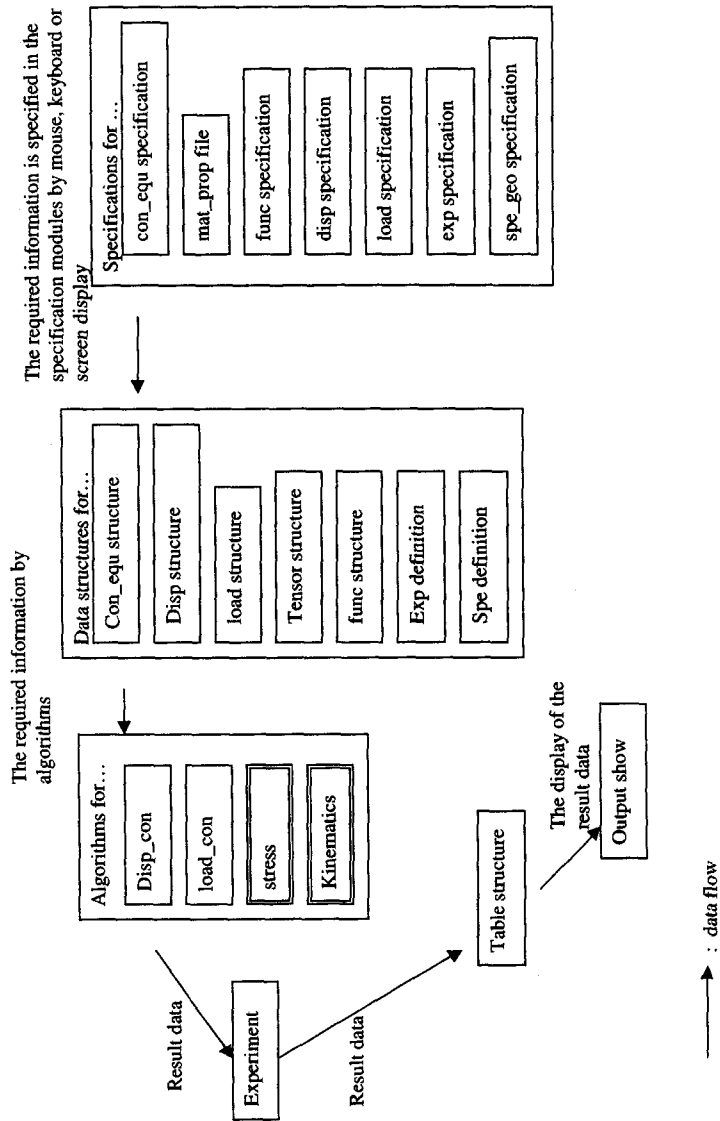


Figure 3.5: The data flow chart for an experiment

As a result of modularity, a clear hierarchical structure of the *Virlab* system is given and such a layered architecture makes the responsibilities and functions of the modules clear. However, the challenge still remains of how to identify the components from the module decomposition.

3.4 Identifying the components

People should be careful when they divide a design into components. Identifying a component depends on many different aspects. Components can be classified as units of analysis, units of abstraction, units of compilation, units of maintenance and units of system management [Szy99]. To identify the components used in the *Virlab*, it is necessary to do a careful analysis.

Figure 3.6 shows that the *Virlab* system is separated as five layers on the basis of responsibilities and functionalities of modules in the system. This figure has the same information as Figure 3.4 except that Figure 3.6 has four dotted lines that are used to split the whole picture into five layers. Table 3.1 shows the composition of each layer and also summarizes the commonalities in each layer.

Based on the Figure 3.6 and summaries of the Table 3.1, the second layer serves as an interface between the *Virlab* software and input devices of the system to facilitate the receiving of the user inputs. The *mouse module* and *keyboard module* work as virtual devices to help the user input the required information needed by an experiment, the *screen display module* can be thought of as external components, such as buttons, combo list, work area, icon, *etc.* that make up the system graphics interface. Therefore, the second layer can be understood as a part of the

application, a graphics user interface built with the external components such as buttons, lists, *etc.* The third layer is all data structures that are fundamental and mandatory for the *Virlab* and are shared by the modules in the fourth layer. Due to its importance and because it is shared by others, all the modules in the third layer should be encapsulated into one component, which is named the structure component. This component is categorized as a fundamental component. In the fourth layer, *stress module* serves to calculate the stress; *kinematics module* serves to obtain the kinematics quantities; *displacement constitutive calculation module* provides the numerical approximation of the constitutive equation to obtain the stress and to recalculate the kinematics quantities when the constitutive equation is introduced in the displacement-controlled experiment, as well as the load constitutive calculation in the load-controlled experiment. So each of the four modules in the fourth layer implements one function of the *Virlab* system. For easy management, it is better to encapsulate each module into one component, they are named as stress component, kinematics component, disp_con component and load_con component, respectively. Since these components emphasize the system function, they are classified as functional components. Modules in the fifth layer such as *importing constitutive equation module* and *experiment module* can be understood as concrete applications from the role of the modules in the system, but *table structure module* is specially designed to store the data in a table form and show *output module* is to display data as a graph. Therefore these two modules are grouped into application component and are respectively named output component and table component.

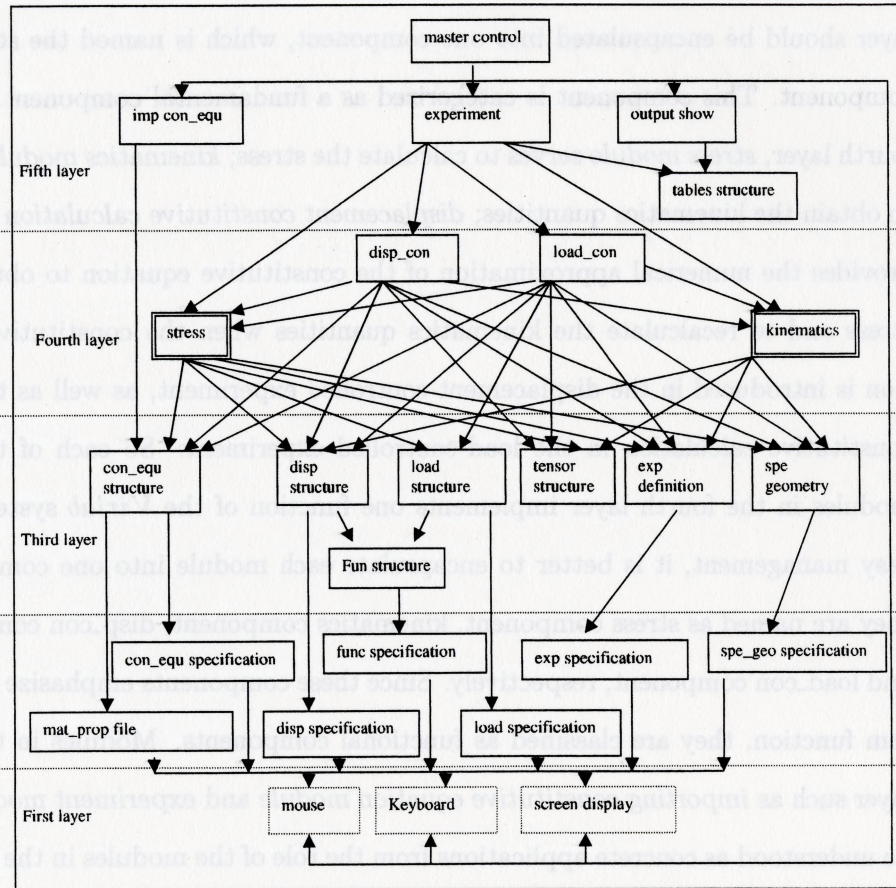


Figure 3.6: System architectural layer

No.Layer	Composition	Similarities
First	mouse, keyboard, screen display	Modules of first layer are provided by the operating system or the developing environment; that is, they are from the external environment
Second	Con.equ specification, func specification, exp specification, disp specification, load specification, spe_geo specification	The purpose of these modules is to query the user to specify the information that is required when the experiment is conducted.
Third	con.equ structure, disp structure, load structure, func structure tensor structure, exp definition, spe geometry	In the modules of the third layer some data structures are designed to represent the information that is needed by the experiment.

Continued on next page

Continued from previous page

No.Layer	Composition	Similarities
Four	stress, disp_con, load_con, kinematics	In the modules of the fourth layer some algorithms are adopted to obtain the displacement, kinematic quantities and stress, which are the experimental results.
Fifth	imp con_equ, experiment, output show, tables structure	In the fifth layer, some modules such as <i>experiment module</i> and <i>importing constitutive equation module</i> implement the concrete application and some modules such as <i>table structure module</i> and <i>output module</i> serve for the concrete application based on the modules of the fourth layer. The secret of the <i>table structure module</i> is one kind of data structure that is specially designed to store the experimental data in the form of a table. <i>Show output module</i> is used to display the experimental results.

Table 3.1: The comparison between the five layers in the *Virlab*

So far in the *Virlab* there are seven components: structure component, stress component, kinematics component, disp_con component, load_con component, table component and output component. From the view of the system these components in the *Virlab* are categorized as fundamental components, functional components and application components. Table 3.2 shows the category of component

Fundamental component	Functional component	Application component
structure component	stress component kinematics component disp_con component load_con component	table component

Table 3.2: Component categories in the *Virlab*

in the *Virlab* and the composition of each component.

3.5 Component-based system architecture

Once modularity of the *Virlab* has been established, it is natural to migrate parts of the *Virlab* system to components by applying the principle of separation of concerns. Now a new challenge is to construct the system with the available components.

Figure 3.7 shows the component architecture of the *Virlab* system. In this figure, the arrow pointing from the concrete application circle to the external component, which could be a visual programming language such as Visual Basic, means that the application uses external components such as buttons. The outer circle represents the concrete application and the inner circles represent each type of component, such as a fundamental component, a functional component or an application component. The parts in the circle separated by a line mean each constitutive component in each type. The ellipsis (...) in the figure means that potential components might be added in the future. In this figure, the structure component, as a fundamental component, lies in the innermost circle of the cyclical layers, since it is necessary and pivotal to all others. The functional components

adopt algorithms to implement different functions such as the calculation of the kinematics quantities on the basis of the information represented with the data structure from the fundamental component. Therefore, functional components are placed on the second innermost cyclical layer.

The application component such as table component and show output component are placed between the outer circle and functional component circle because they work as a bridge between the application and functional components. For example, the table component is designed because the table form is needed from the requirements of the application. The outer circle is for the real application. The real application is based on the systematic components such as the structure component and the external components such as buttons that make up the application's graphics interface. In the real application the communications among the components are finished by the component interfaces. In this figure, from the view of the systematic structures each layer is isolated from others and the *Virlab* is built from the inner circle to the outer circle. However from the view of data flow, data is input from the concrete application's graphical interface to the inner circle (structure component) and then flows from the inner circle to outer circle and finally the experimental results are obtained from the concrete application as the graphical output interface. Regardless of different views, communications among components and between components and real applications are furnished by the components' interfaces. Therefore, the design for the *Virlab* system is called a component-based design.

Upon the completion of the component architecture of the *Virlab*, the advantages of the component-based architecture are clear.

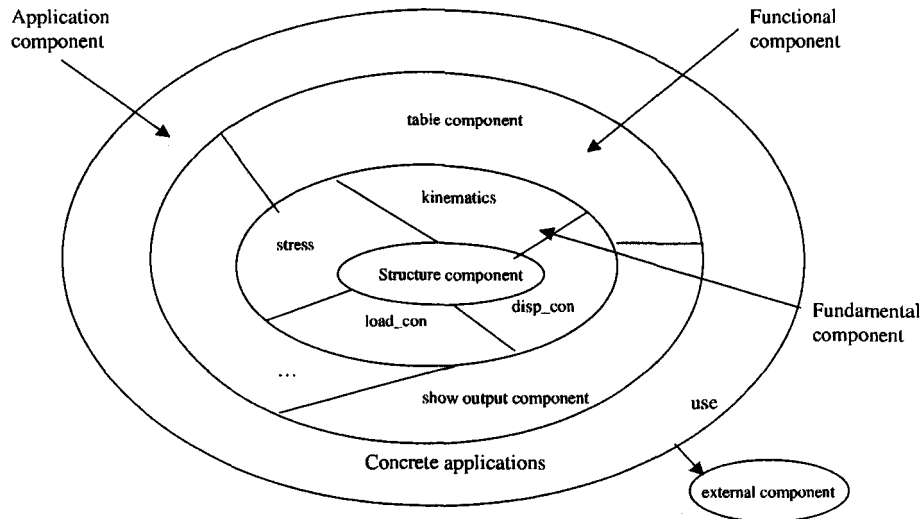


Figure 3.7: Component architecture of the *Virlab* system

- Helpful to understand

Although the *Virlab* system is a large and complex system, Figure 3.7 makes this large and complex system easy to understand by presenting it at an abstract level.

- Valuable to construct

The component architecture and architectural description works as a partial blueprint for the development by showing the major components and their relationships. It will be helpful to guide designers to implement the whole system because it will act as a reference document and it is also useful for newcomers to the team to quickly grasp the system architecture.

- Useful for analysis and management

Clear architecture is always useful for maintainers to help them analyze and

manage the components.

- Support rapid development

Clear architecture supports a rapid development. The second circle in Figure 3.7 is for the functional components. The designer can freely choose which components will be used in the version based on different functionalities required by the system. For example, assuming the four functional components are available, a system that does only load-controlled experiments could be quickly created by removing the `disp_con` component.

So far we have devoted considerable effort in giving a definition of the component and the reasons that we need a clear definition of the component, identifying a component and constructing a component-based architecture; however, nobody but the authors can understand the system without adequate documentation. Furthermore the authors themselves might forget their design as time goes by. Therefore, specification documentation is necessary for the *Virlab* system.

3.6 Documentation

Documentation plays an important role in the software development. Therefore documentation should be written and maintained from the beginning and throughout the lifetime of a system. This section includes three documents for the *Virlab* system. Each document is created as the specific area of the *Virlab* design is being molded. Since the modularity is a precondition of the component, Section 3.6.1 documents the modules that each component is built from. The communications among the modules happen through the interfaces. Therefore, Section 3.6.2 is

the module interface specification, where the interface of each module identified is specified. Finally, Section 3.6.3 is the documentation of the components identified in the previous section, based on the documentation of the module and module interface specification.

3.6.1 Documenting modules

We followed the principle of the information hiding and divided the system into modules by the use of system decomposition with the goal of a hierarchical structure as described in the section 3.3. In this section, we will look into a document that records the system architecture that is known as a module guide [PCW85].

The module guide for the *Virlab* is attached in Appendix C. The purpose of this module guide is to make the design for the *Virlab* system explicit, which will facilitate changes to the system in the future. Table 3.3 is a module guide for the *experiment definition module* and shows an example on how to document a module in the *Virlab*. The template for documenting modules is based on the module structure of A-7E flight software by the Naval Research Laboratory [PCW85].

In the module guide for the *Virlab*, the module's secret is used as the main description that characterizes each module. This part typically embodies the notion "one module one secret". The service of the module is intended to give some hints that this module implements a particular aspect of the system. In addition to its secret and service, expected changes are added to help the reader to predict and understand the possible changes in the future. The entry for "prefix" is used to trace the implementation of the module in the *Virlab*. "Prefix" is a

Module Name	Experiment definition module
Module service	This module provides experiment configuration information.
Module secret	A data structure to represent the experiment configuration. Secret type: software design.
Expected changes	Since experiment configuration is represented with a data structure, the same experiment might be represented with other data structures depending on the designers' choice.
Prefix	ed_

Table 3.3: *Experiment definition module*

short string that will be prepended to all the access routines belonging to the same modules, so that the conflict of the names among the access routines in the different modules can be avoided. If a module does not have a prefix, this means the operating system and/or development environment provides the needed functionalities.

To make the module guide of *Virlab* complete and easy to understand, it also describes the system decomposition. Since each module hides some design decision of the system, generally the hidden information of each module can be divided into the following three classes: behavior-hiding module, software decision-hiding module and hardware-hiding module [PCW85]. The diagram is shown in the Figure 3.1. The module guide for the *Virlab* also comprises the diagrams of the use relation between the modules (see the Figure 3.4). To this point in the document, the followings have been developed: a complete module guide, which includes the system decomposition, use relation of the modules and detailed

module description.

The effort devoted to creating a module guide was significant. However, the effort is worthwhile because a module guide provides the following benefits:

- Allows verification of the system

After the module guide has been finished, it is possible to check for various errors, to discover the possible inconsistencies, to review the feasibility of the decomposition and to evaluate the flexibility of the design.

- Allows one to quickly grasp the system [PCW85]

Since the module guide shows the clear responsibilities among the modules, it is helpful for newcomers to the project team to understand the system.

- Removes ambiguity

In our experience of conceiving and writing the module guide for the *Virlab* we went through an iterative thinking process of identifying the secrets and clarifying the responsibility of each module and the relationship between the modules. As an example of this process, the module guide in the first version included both an *engineering strain tensor module* and a *lagrangian infinitesimal strain tensor module*, but we chose one of them in the current version because they had the same responsibility when we compared the services they provide. As a result, ambiguities in the design were discovered through the process of writing the module guide.

- Provide guidance [Par72]

The existence of the module guide not only benefits ourselves it also will provide support and guidance for programmers and maintainers.

3.6.2 Documenting the module interface specification

A significant advantage of the module design is that it allows parallel development of the system among programmers. However the module guide for the system is not enough, the module interface specification (MIS) is also necessary for the system to record the crucial design. “The module specification tells you both how to use that module and what that module must do [PCW85]”. The purpose of a MIS is to describe how each module is to be used, for example, how can a programmer code interfaces with the module? What will the access routines require for the inputs and what will they return for the output? A MIS for the *Virlab* system is provided in Appendix D. In this section we will first take a close look at the content and then we will discuss our design decisions for the MIS for the *Virlab*.

Contents in the MIS

The following shows a MIS for “the *constitutive equation structure module*”. We use it to illustrate the contents for the MIS. Generally a MIS for each module includes two sections: one section for interface syntax and the other section for interface semantics [HS95].

Constitutive equation structure module

Prefix: cs_

Reference: MG - Section C. 3.2.1

Interface syntax

Imported data type:

```
PropertyListT from the material property file module
PropertiesT = tuple of {
  propertyname: string
  propertyvalue: real
}
```


PropertyValueT from the material property file module

PropertyValueT = sequence of real

LISTNUM from the tensor data definition module

LISTNUM 15

Exported data type:

ConstitutiveEquationT = tuple of {

name: string

the_number_of_material_properties: integer

material_properties_list: PropertyList

deformation_list: DeformationListT

}

DeformationListT = sequence [LISTNUM] of boolean

Exported constant: none

Exported function:

Routine names	Inputs	Outputs	Exceptions
cs_g_constitutiveequation		ConstitutiveEquationT	
cs_s_constitutiveequation	string DeformationListT PropertyValueT		
ce_g_writetofile	ConstitutiveEquationT		failure_to_open
ce_s_readfromfile	string	ConstitutiveEquationT	failure_to_open file_not_exist

External functions:

mpf_g_numberofproperties from the material property file module

mpf_g_propertyname from the material property file module

file Openfile(filename: string)

Open a file whose name is filename, if return value is zeron, opening a file is successful, otherwise failure

Readfile(f: file, type: string)

Read the value of the specified type from an current position of a file pointer and return this value

Writefile(f: file, var: string)

Write the value of a variable whose type is type constructor into the file

Interface semantics

State variable:

con: ConstitutiveEquationT

f: file

Local variable: local: PropertyListT

State invariant: none

Assumption:

- Functions `cs_g_readfile` and `cs_g_writetofile` are used to operate the file in which the constitutive equation is saved. Actually the order of the format is irrelevant but it should be consistent so that the order of information on the constitutive equation that is read from the file is the same as the order that is written to the file.

Access routine semantics:

`cs_s_constitutiveequation(name:string, deform:DeformationListT,
value:PropertyValueT)`

Exception: none

Transition: `con.name := filename`
`con.the_number_of_material_properties := mpf.g_numberofproperties(name)`
`local := mpf.g_propertyname(name)`
`local := mpf.sg_propertyvalue(value)`
`con.material_property_list := local`
`mpf.g_propertyname(name)`
`con.deformation_list := deform`

`cs_g_constitutiveequation()`

Exception: none

Output: `out := con`

`cs_g_readfromfile(filename:string)`

Exception: (`Openfile(filename)` is `successful` \Rightarrow `fail_to_open`,
`file_not_exist`)

Transition/output: `f := Openfile(filename)`
`local.name := filename`
`local.the_number_of_material_properties := Readfile(f, "integer")`
for `i=0` to `local.the_number_of_material_properties`
`local.material_property_list[i] := Readfile(f, "PropertyT")`
`local.deformation_list := Readfile(f, "DeformationlistT")`
`out := local`

`cs_g_writetofile(conequ:ConstitutiveEquationT)`

Exception: `Openfile(conequ.name)` is unsuccessful \Rightarrow `fail_to_open`

Transition/output: `f:=Openfile(conequ.name)`
`Writefile(f,conequ.the_number_of_material_properties)`
for `i=0` to `conequ.the_number_of_material_properties`
`Writefile(f,conequ.material_property_list[i])`
`Writefile(f,conequ.deformation_list)`

Comments:

- `DeformationListT` type is a sequence of boolean. This type is used to present what deformation definitions are used in the constitutive equation.

- Interface syntax section

The interface syntax section includes the declaration of the exported data types, functions names, parameters and return types of the functions, exceptions in the functions, exported constants, imported data types, environmental variables and external functions. However, among all the attributes listed above, because exported data types, exported constants and exported functions directly show the services of the module, they are listed as the core part for the module, regardless of whether a module has any exported data type or exported constants or not. The other attributes will be included only when they are needed. The exported functions as interfaces are called when outside modules or programs use this module. The exported data types and exported constants are listed as new data types provided by this module. The external functions mean that these functions are needed in this module and are assumed to be provided by other modules in the system or the development environment.

- Interface semantics section

The interface semantics section mainly focuses on how exported functions in a module are called and what each does and/or returns based on the inputs. It includes the basic parts such as state variables, state invariants, assumptions and access routines semantics and the additional parts such as local variables, local functions, event tables and comments. State variables are internal to the module and are shared by the members in the module. State invariants are the predicates that remain true after the successful execution of any access program. Assumptions are the preconditions that a module requires its users to meet. Access routine semantics explain the details on exported functions. For each function, the possible conditions for the exceptions are specified in the heading “exception”. How a state variable or an environmental variable is changed is explained in the heading “transition”. The outputs that the exported function returns are specified in the heading “output”. The event table is helpful to understand the connections between the events and exported functions. The comments are listed based on some design considerations and explanations of the details. Local variables and local functions are used in the cases where there is a complex specification.

Some design decisions for the MIS of the Virlab

- A section for conventions used in the MIS

To avoid the ambiguity for the readers, we list a section for the conventions we use to document the MIS.

- Use as precise a description as possible

$$class=H_1 \cap type=H_2$$

		H_1		
		uniaxial	biaxial	multiaxial
H_2	displacement -controlled	ds_s_initdisp ds_s_disputype	ds_s_initdisp ds_s_disputype ds_s_dispvtype	ds_s_initdisp ds_s_disputype ds_s_dispvtype ds_s_dispwttype
	load-controlled	ds_s_initdisp	ds_s_initdisp	ds_s_initdisp

Table 3.4: The order for the initialization

We advocate the formal specification indicated by Parnas. For this goal we try our best to use mathematics and Parnas tables [JPZ97] in our MIS to make the description more precise and easy to follow. Table 3.4 is an example of a Parnas table used in the MIS for the *displacement specification module*. It is more obvious to specify the initialization process for the displacement under the different conditions in a table than in prose. Also, we adopt predicts for the mathematical expressions to make the description unambiguous.

- The prefix of a module

In the module guide we mention that the modules with prefixes will be implemented so those modules are further specified in the MIS. The same prefix is found below the name of the corresponding module at the beginning of the specification for each module. The purpose of a prefix is for quick reference among modules. For example, the *displacement structure module* uses the function `fs_g_functionvalue()` to calculate the displacement function. This function is declared in the external functions part of the syntax section of the *displacement structure module*. From its prefix “fs” we know that

this function is implemented in the *function structure module*.

- The usage of reference

The reference is found next to the prefix part in the specification for each module. Since the interfaces of the modules are abstracted from the module guide, it is helpful to point to the source for the specification. A section number from the module guide where this module is introduced is listed after the heading “reference”. From this section number, the corresponding section in the module guide can be quickly located and then the services and secrets of the module can be reviewed.

- @ in the event table

Event tables were introduced to specify the software requirements for the A-7E [Hen80]. The notation @T(condition₁) is used to denote the occurrence of condition₁ becoming true. Event tables show when certain functions should be performed or when periodic functions should be started or stopped. Table 3.5 shows an example for the event table used in our MIS. In this event table the notation @Click(buttonname) is used to denote the occurrence of the event that the button ‘buttonname’ is clicked. For example, an event @Click(Confirm) happens when the ‘Confirm’ button is pressed. From Table 3.5 we know that when the event @Click(Confirm) happens, functions `ed_s_constitutiveswitch`, `ce_s_constitutiveequation`, `mpf_g_numberofproperties`, `mpf_g_propertyname` and `mpf_sg_propertyvalue` should be called.

- Developer/designer/user

At the beginning of the design for the *Virlab*, we tried to summarize a

Condition	Event	Action
When the 'Confirm' button is pressed	@Click(Confirm)	ed_s_constitutiveswitch, ce_s_constitutiveequation, mpf_g_numberofproperties, mpf_g_propertyname, mpf_sg_propertyvalue
When the 'Save' button is pressed	@Click(Save)	ce_g_writetofile
When the Cancel button is pressed	@Click(Cancel)	Do nothing

Table 3.5: An example for an event table

unique form to describe the constitutive equations and then numerically approximate this form. We realized that it is difficult to do this because of the complexity and diversity of constitutive equations. To make our design feasible we decided to move some of the responsibilities for the design outside of our system. To make this notion concrete we defined three roles for people interacting with the system: the developer, the designer and the user. A user refers to a person who will do the experiment with *Virlab* software, e.g. a student. A designer is a person who designs an experiment with the *Virlab* software for a user to do an experiment or for himself to do research. A designer is required to have solid professional knowledge about what a material experiment needs. Moreover, the designer should have basic software engineering knowledge because he/she can add new materials for the experiment into the *Virlab* based on the experiment requirement, which requires that he/she should understand the documentation for the module guide and

module interface specification. A developer is a person who develops and/or extends the *Virlab* software by implementing new functions of the *Virlab* software or enriching the system on the basis of the *Virlab* blueprint. A developer should have strong software engineering knowledge because he or she has the responsibilities to build and update the documentation for the *Virlab*.

- The usage of the term ‘virtual’

The term ‘virtual’ is used in the object-oriented programming language and means if the method is defined as virtual in the base class, its concrete implementation will be provided in the subclass of this base class. Because the term ‘virtual’ is used clearly to delegate the class’s responsibility for the implementation, it is borrowed into our MIS to define the attribute of the access programs in some of the module interface specifications. For example, in the *displacement constitutive (disp_con) calculation module*, the exported function `dcc_disconstitutive()` is defined as virtual, which means that the designers have responsibility to provide the implementation of the corresponding access programs. This allows us to clearly show what aspects of the system are the responsibility of later developers.

Theoretically after a MIS is written for a module, the first step toward implementation is to prepare a module internal design (MID). A MID deals with the concrete state of the module [HS95], *e.g.*, the internal design for abstract data types. However, in the *Virlab* system a MID was not explicitly created because *Virlab* is not a complicated system and the mapping between the abstract state and the concrete state is straightforward. When we designed the MIS, we chose

those simple and obvious data types that have direct counterpart in most programming languages. For example, a tuple can be implemented as a structure in the C language. Based on these consideration, we our some efforts on designing the MIS. Therefore, the fact that we do not have an explicit MID will not affect the implementation of *Virlab* later.

3.6.3 Documenting the component description

In Section 3.5 we concluded that the design for the *Virlab* system is a component-based design. Since the components are derived from the modularity of the system, documenting the module guide and module interface specification for the *Virlab* makes documenting the components easier. The purpose of the component descriptions is to describe what a component can provide, how to use the component together with the MIS, the role of the component in the system and the source of the component. The component description for the *Virlab* is attached as Appendix E.

Since the goal of documenting the component is to make what it requires and what it provides explicit, the component description is composed with the basic parts such as name, role in the system, service, composition and interface specification and an additional part for comments. The “service” part generally specifies what the component provides and it is also used to give directions for the production of the component interface. The “interface specification” part is designed to specify what it requires in detail. The “role in the system” part is used to denote the component’s role in the system so that newcomers in the team can quickly grasp the composition of the system. The “composition” part

is used to specify the source of the component because the characteristic of the component we use in the *Virlab* is module-based. The “comment” part is designed to supplement the source of data types and/or constants occurring in the interface specification.

Name	Kinematics component
Role in the system	Functional component
Service	Based on the definitions given by [Maz70], Kinematics component is used to calculate the kinematics quantities.
Composition	material deformation gradient module spatial deformation gradient module material displacement gradient module spatial displacement gradient module Cauchy’s deformation tensor module Green’s deformation tensor module Lagrangian finite strain tensor module Eulerian finite strain tensor module Lagrangian infinitesimal strain tensor module Eulerian infinitesimal strain tensor module True strain tensor module Stretch tensor module Stretch ratio tensor module

Continued on next page

Continued from previous page

Name	Kinematics component
Interface specification	<p>The composed modules' interfaces are available for the use. Please refer to the corresponding module interface specification found in the Appendix D. Based on the similarities among all the composed modules' interfaces in the kinematics component, two interfaces are summarized below.</p> <ul style="list-style-type: none"> • <code>kc_knownquantity(comeflag, outflag: TensorFlagT, kq: TensorDataT)</code> <p>Exception: exceptions are triggered from the calling programs and same as the exceptions from the calling programs</p> <p>Output: output is based on the value of outflag shown in the Table3.7</p> <ul style="list-style-type: none"> • <code>kc_geometry(outflag: TensorFlagT, disp: DisplacementT, sg: SpecimenGeometryT)</code> <p>Exception: exceptions are triggered from the calling programs</p> <p>Output: output is based on the value of outflag shown in the Table3.8</p>

Continued on next page

Continued from previous page

Name	Kinematics component
Comments	<ul style="list-style-type: none"> • TensorFlagT is from the Tensor data definition module in the structure component • TensorDataT is from tensor data definition module in the structure component • DisplacementT is from the displacement structure module in the structure component • SpecimenGeometryT is from the specimen geometry module in the structure component • Capital letters such as MDPG are from the tensor data definition module in the structure component

Table 3.6: A component description for the kinematics component

Table 3.6 shows a component description for the kinematics component as an example. In this table it is easy to understand the “name”, “role in the system”, “service”, “composition” and “comments” parts for the kinematics component. In the “interface specification” part, it is explained that interfaces of composed modules in kinematics component are the kinematics component’s interfaces and

outflag=MDG	mdg_g_knownquantity(kq, comeflag)
out=SDG	sdg_g_knownquantity(kq, comeflag)
outflag=MDPG	mdpg_g_knownquantity(kq, comeflag)
outflag=SDPG	sdpg_g_knownquantity(kq, comeflag)
outflag=CDT	cdt_g_knownquantity(kq, comeflag)
outflag=GDT	gdt_g_knownquantity(kq, comeflag)
outflag=LFST	lfst_g_knownquantity(kq, comeflag)
outflag=EFST	efst_g_knownquantity(kq, comeflag)
outflag=LIST	list_g_knownquantity(kq, comeflag)
outflag=EIST	eist_g_knownquantity(kq, comeflag)
outflag=TST	tst_g_knownquantity(kq, comeflag)

Table 3.7: Output for kc_knownquantity

outflag=MDG	mdg_g_geometry(kq, comeflag)
out=SDG	sdg_g_geometry(kq, comeflag)
outflag=MDPG	mdpg_g_geometry(kq, comeflag)
outflag=SDPG	sdpg_g_geometry(kq, comeflag)
outflag=CDT	cdt_g_geometry(kq, comeflag)
outflag=GDT	gdt_g_geometry(kq, comeflag)
outflag=LFST	lfst_g_geometry(kq, comeflag)
outflag=EFST	efst_g_geometry(kq, comeflag)
outflag=LIST	list_g_geometry(kq, comeflag)
outflag=EIST	eist_g_geometry(kq, comeflag)
outflag=TST	tst_g_geometry(kq, comeflag)

Table 3.8: Output for kc_geometry

two interfaces for the kinematics component are summarized based on the similarities of known interfaces. The semantics for the summarized interface are included with the output part and exception part. In the Table 3.6, if the required parameter `outflag` for `kc_knownquantity` function is `MDG`, then `kc_knownquantity` will use `mdg_g_knownquantity` function. Simultaneously required parameters `comeflag` and `kq` are passed into `mdg_g_knownquantity` function. So for this case the output of `kc_knownquantity` function is the same as the output of `mdg_g_knownquantity`. Also the exceptions for `kc_knownquantity` function are from the exceptions for `mdg_g_knownquantity`. The details on `mdg_g_knownquantity` can be found in a MIS for *material deformation gradient module* in the Appendix D. It is obvious that the component descriptions are closely bonded with the MIS in the *Virlab*. Therefore the component descriptions should always be used together with the MIS in the *Virlab* for programmers and maintainers.

Chapter 4

An overview of the *Virlab* software

In Chapter 2 we described the testing of real materials. Chapter 3 gave the component-based design of the *Virlab* software by dividing the system into several components such as the fundamental component, the functional components and the application components. Based on the design of the *Virlab*, the graphical user interface of the *Virlab* was implemented with Visual Basic 6.0 and the constituent components of the *Virlab* were implemented with Visual C++ 6.0. The components followed the COM [Rog97] standard that specifies how to build components.

In this Chapter, we provide an overview of the *Virlab* software. Section 4.1 gives the general function descriptions of the *Virlab* software. Section 4.2 describes how to use *Virlab* software to do a uniaxial displacement-controlled experiment for a viscoelastic material, which was previously described as a case study in Chapter 2.

4.1 An introduction to the *Virlab* software

The *Virlab* was designed and developed as a virtual laboratory, an open and flexible software environment that is used to simulate a set of experiments using a computer. The *Virlab* software has the following attributes:

- Friendly user interface

From the visual point of view, the *Virlab* software is composed of several windows and each window includes several buttons, text information and possibly some smaller windows. The captions for the windows and buttons explains their purpose. The intention is that the system will be easy for new users to use in a short amount of time. The program also has an information center window and a warning center window to help exchange information between the user and the system. The information center provides instructions for the experiment and the warning center is used to display warnings and error messages from the system.

- Allows the user to select the experiment class and type

The experiment class refers to the displacement-controlled experiment or the load-controlled experiment. The experiment type means a uniaxial experiment, a biaxial experiment or a multiaxial experiment. The experiment classes and types are displayed in the way of radio buttons, which allows the user to easily select the experiment class and type.

- Allow the user to setup the configuration for the experiment

The *Virlab* gives the user freedom to configure the information that the chosen experiment requires. For example, the user specifies the following:

- The type and coefficients of the function that is used to describe the independent variables for the load or the displacement controlled experiment
 - Test specimen geometry
 - Material properties
 - Constitutive equation
 - Time configuration
- Output the results of the experiment

The *Virlab* allows the user to output the results of the experiment. The experimental data is displayed in tabular form in the *Virlab*. Based on columns the user selects, the experimental data can also be plotted as a curve. This allows the user to conveniently analyze and compare experimental data.

The details of the system are provided below as follows: Section 4.1.1 describes the main window of the *Virlab* software. Section 4.1.2 explains the setup window and Section 4.1.3 illustrates the output window.

4.1.1 The main window

To invoke the main window of *Virlab*, double click *Virlab* on the Windows environment. After copyright information is shown, the window for the experiment selection is displayed, as shown in Figure 4.1. In this window the user can select the experiment type and the experiment class. When the experiment class and type are selected, the user can click the Next button to continue the experiment. At any time, the use can quit the *Virlab* by clicking the Exit button.

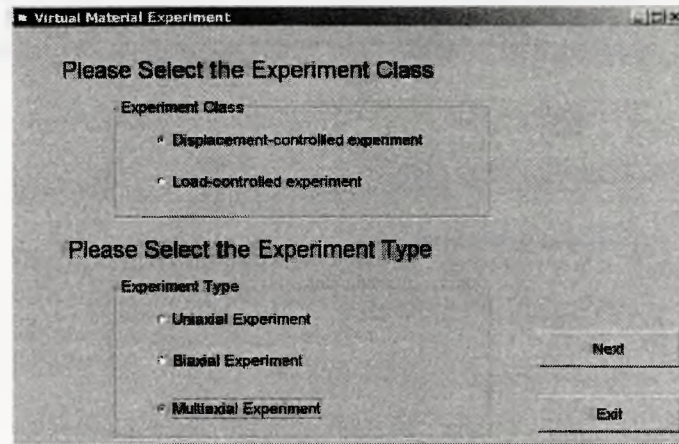


Figure 4.1: The window for the experiment type selection

Once the experiment class and type are selected and the Next button is clicked, the experiment window shown in Figure 4.2 will come up. The experiment window is composed of three small windows and several buttons. The picture window shown in the upper left corner of the experiment window is used to intuitively display a picture for the chosen experiment. The information center window shown in the lower left corner is used to display the experiment instructions. For example, in the Figure 4.2, the user is told to set up the experiment first. The warning center window shown in the middle of the experiment window is used to display the error messages. To invoke the setup window, the user clicks the Setup button.

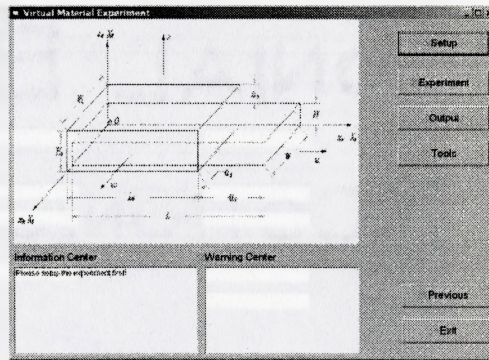


Figure 4.2: Experiment window

4.1.2 The setup window

The setup window shown in Figure 4.3 is used to configure the experiment and to setup the related information required by the experiment. To specify the function type and function's coefficients, that function is used to describe the independent variables for the load or the displacement controlled experiment. The SpecifyFunction window shown in Figure 4.3 should be invoked by clicking the Specifyfunction button. In this window, the specified text and input area will be available only if the corresponding function type button is selected. To specify the specimen geometry, click the SpecifySpecimenGeometry button to invoke the SpecifySpecimenGeometry window shown in Figure 4.4. To specify the time configuration, click the SpecifyTimeSetup button to call the SpecifyTimeSetup window shown in Figure 4.5. To specify material properties and information on the constitutive equation, click the ConstitutiveEquation button to call the ConstitutiveEquation window shown in Figure 4.6. The existing constitutive equations in the *Virlab* will be loaded and ready for display. In this window, shown in Figure 4.6, the user

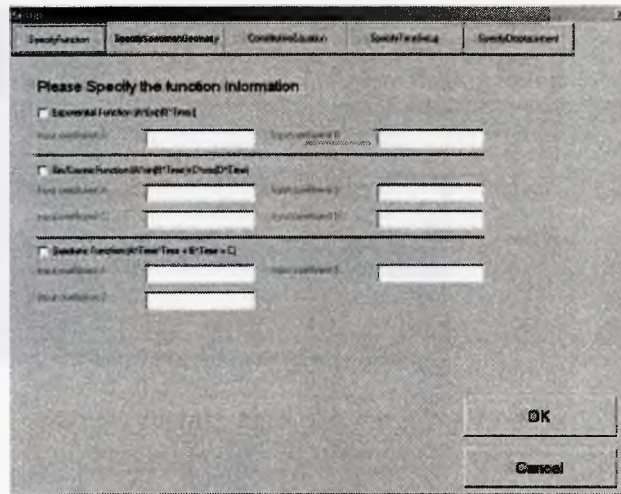


Figure 4.3: Setup window

can also set up material properties for the selected constitutive equations. Based on the specified experiment class and type, the SpecifyDisplacement button or SpecifyLoad button comes up in the Setup window. Click this button to specify the load or displacement function that describes the corresponding independent variable from the available functions shown in Figure 4.7.

4.1.3 The output window

Once the setup for the experiment is finished, the experiment is ready to run by clicking the experiment button on the experiment window shown in Figure 4.2. If there is no error message in the warning center window, the message “Experimental data is ready for output” will be displayed in the information center. The user next clicks on the Output button on the experiment window,

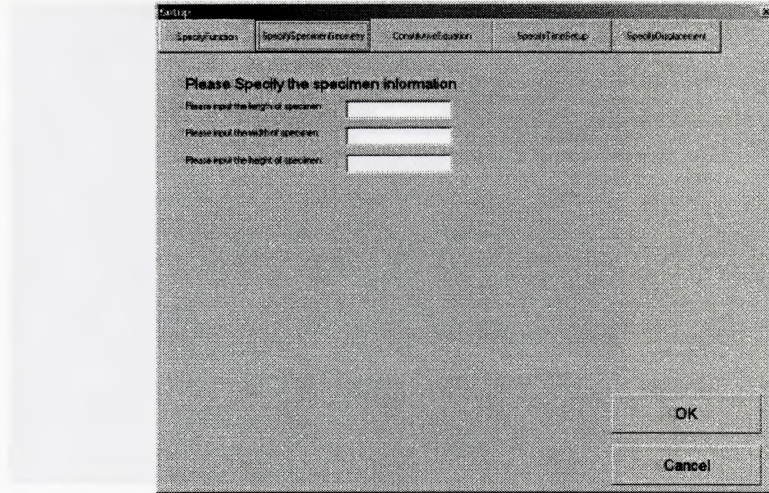


Figure 4.4: The SpecifySpecimenGeometry window

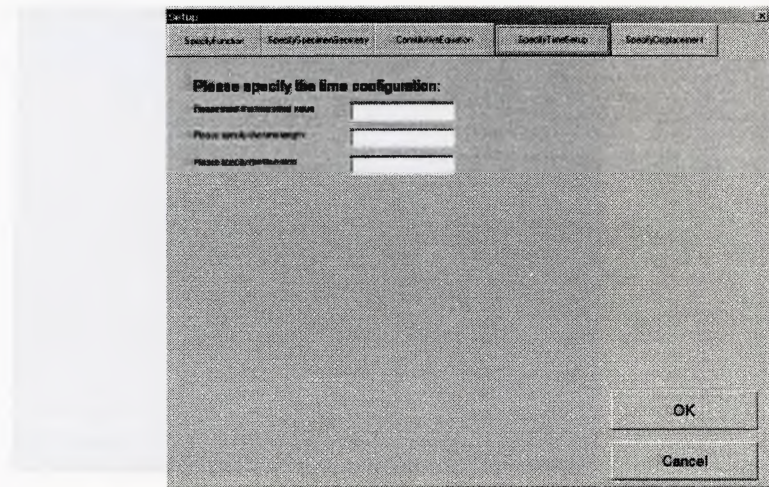


Figure 4.5: The SpecifyTimeSetup window

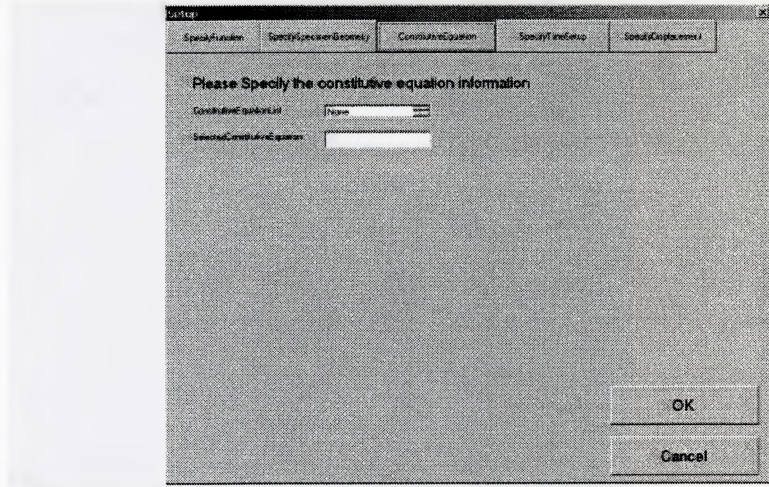


Figure 4.6: The ConstitutiveEquation window

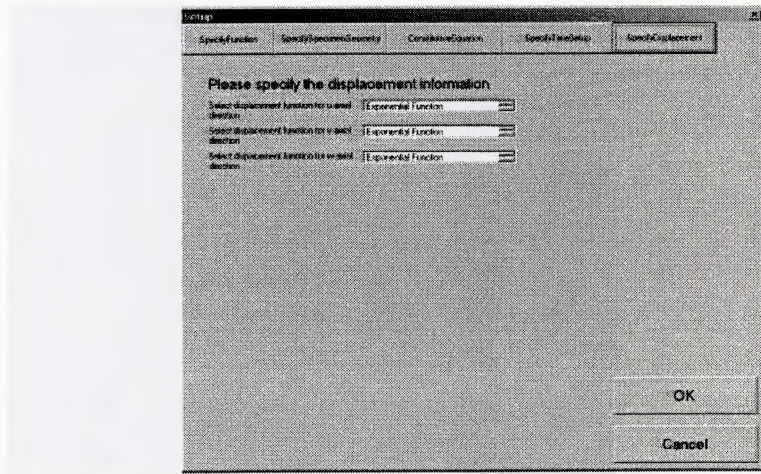


Figure 4.7: The SpecifyLoadOrDisplacement window

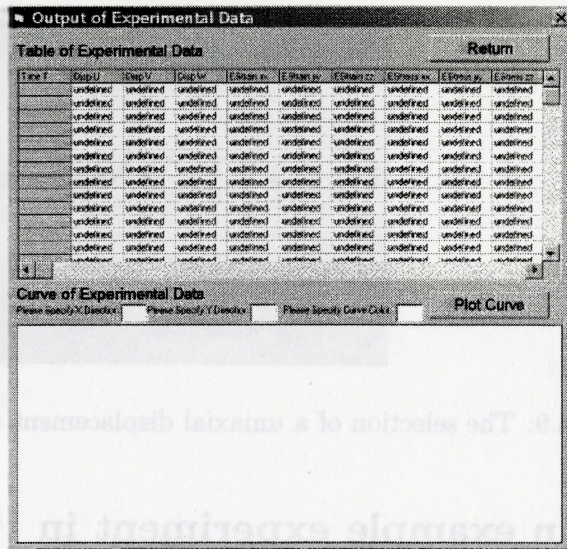


Figure 4.8: The output window

which invokes the output window shown in Figure 4.8. The output window is composed of two subwindows: a table window, the upper window in the Figure 4.8, and a graph plotting window, the lower window in the Figure 4.8. The table window will display the experimental data from the experiment. To plot the graph, the user should specify the sources for the x and y axes, the column numbers for the x and y axes, and the colour of the corresponding curve. When the x , y axes and the colour for the curve are specified, the user clicks the plot button and the corresponding curve will be displayed.

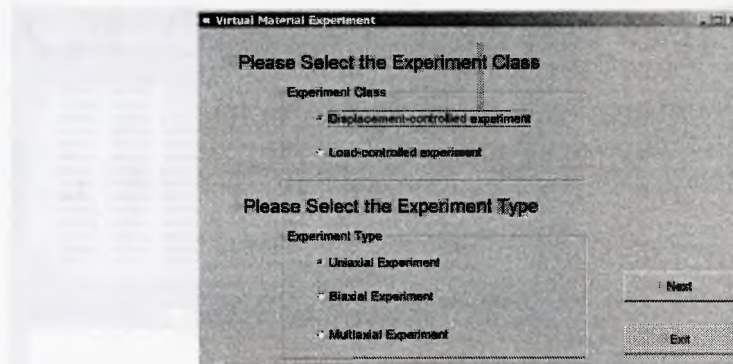


Figure 4.9: The selection of a uniaxial displacement-controlled experiment

4.2 An example experiment in *Virlab* for a uniaxial displacement-controlled experiment for a viscoelastic material

In Section 2.3 of Chapter 2, we presented a uniaxial displacement-controlled experiment as a case study to explain real material testing. In this section, the user follows the step-by-step instructions to use the *Virlab* software to do this experiment.

1. First, select the experiment class and type in the main window shown in Figure 4.9 and then click the Next button and the experiment window shown in Figure 4.10 comes up.
2. Set up the information on this experiment by clicking the Setup button in Figure 4.10 and enter the Setup window shown in Figure 4.11.
3. Check Exponential Function box and then the specified text and input area

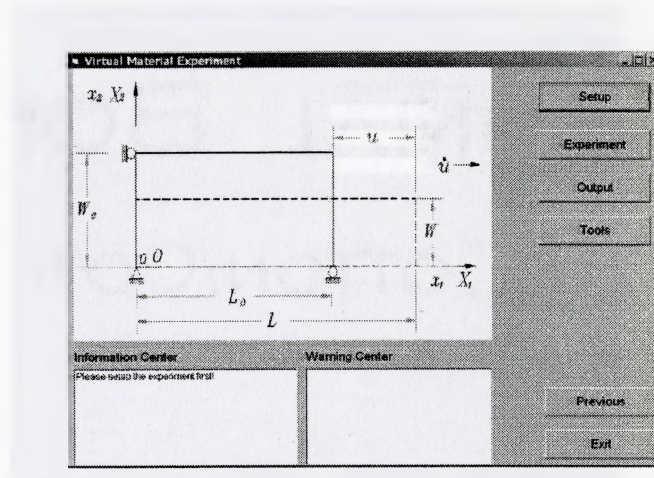


Figure 4.10: Experiment window for a uniaxial displacement-controlled experiment

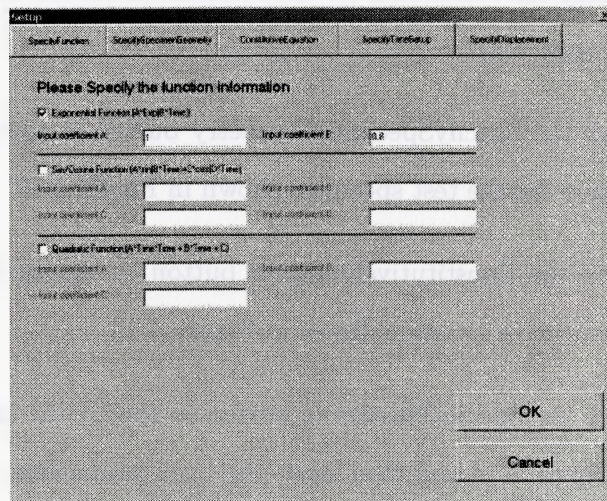
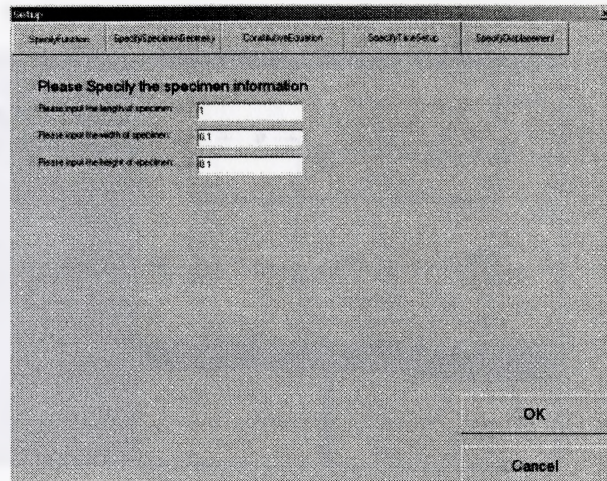


Figure 4.11: Input the function information



Specify

SpecifyFunction SpecifySpecimenGeometry ConstitutiveEquation SpecifyTimeSetup SpecifyOutputFormat

Please Specify the specimen information

Please input the length of specimen: 1

Please input the width of specimen: 0.1

Please input the height of specimen: 0.1

OK

Cancel

Figure 4.12: Input the geometry information

for the exponential function become active.

4. Specify the coefficients for the exponential function in the input area shown in Figure 4.11
5. Click the **SpecifySpecimenGeometry** button to specify the geometry information for the test specimen shown in Figure 4.12.
6. Click the **ConstitutiveEquation** button to specify the information on the constitutive equation shown in Figure 4.13.
7. Click the **SpecifyTimeSetup** button to specify the time configuration shown in Figure 4.14 and then input the time parameters.
8. Click the **SpecifyDisplacement** button to select the required function type shown in Figure 4.15 by clicking the list box.

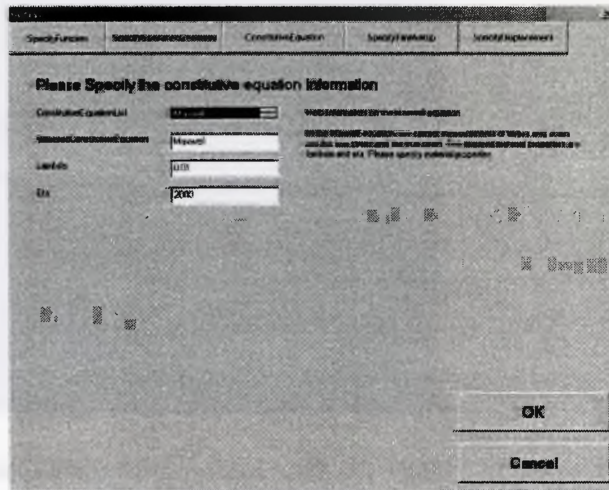


Figure 4.13: Specify the constitutive equation

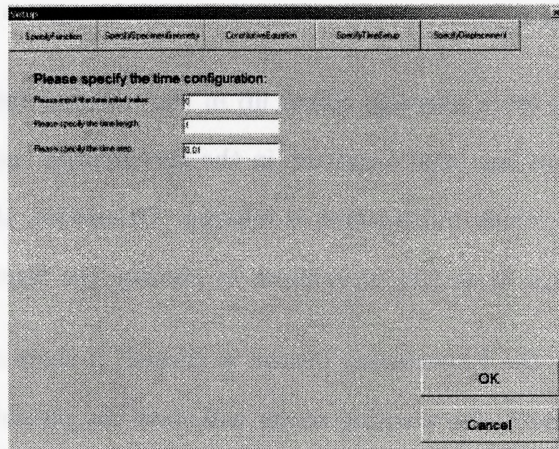


Figure 4.14: Specify the time configuration

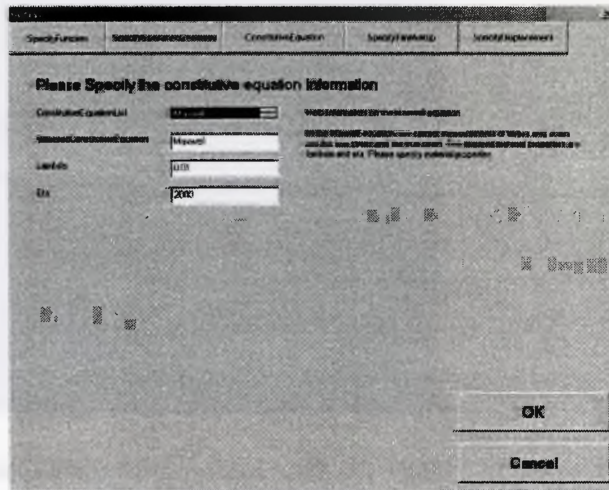


Figure 4.13: Specify the constitutive equation

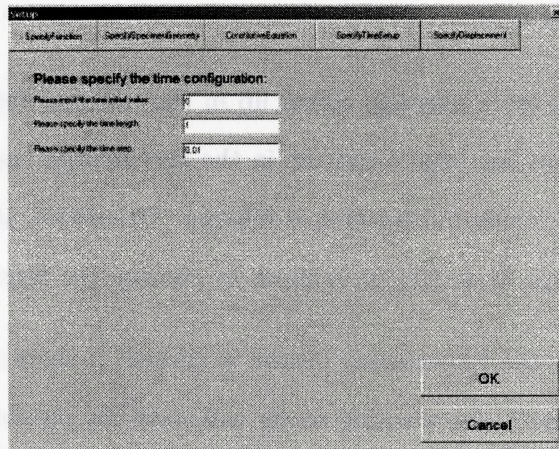


Figure 4.14: Specify the time configuration

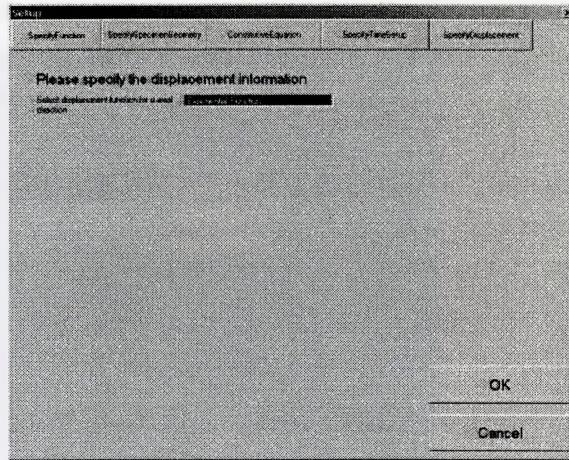


Figure 4.15: Select the function type for displacement variable

From the second step to the seventh step, the order of clicking the heading buttons does not matter. Once all the information is specified, the user clicks the OK button and then returns to the experiment window shown in Figure 4.10.

9. If the error message comes up in the warning center, it means some information has been forgotten in the specification and the *Virlab* will prompt as to which information is missing. Otherwise, the information center will prompt to do the experiment by clicking the Experiment button.
10. After the Experiment button is clicked, if there are some errors in the experiment, the warning center will show the possible reasons for errors. Otherwise, once the experiment is done, the information center will prompt “Experimental data is ready for output”. Now click the Output button and display the experimental data shown in Figure 4.16.

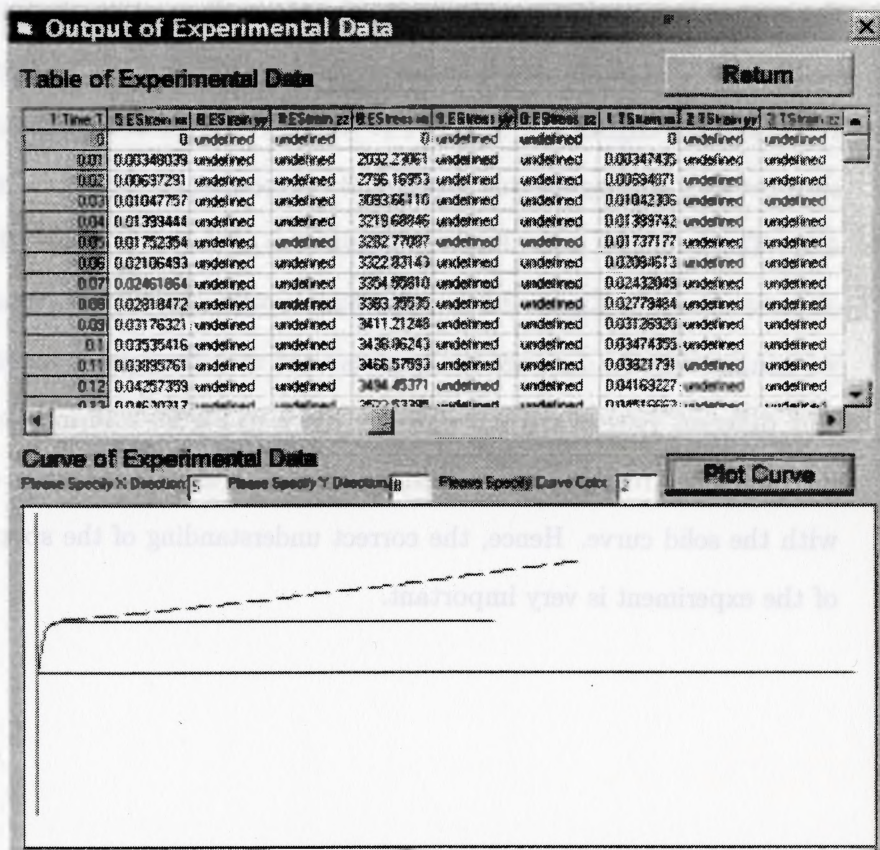


Figure 4.16: The output for the experiment

11. The specification for the constitutive equation “Maxwell equation” shown in Figure 4.13 clearly shows that true stress and strain are required. Therefore, the user plots the relationship between true stress and true strain shown as a solid curve in Figure 4.16. A user may wonder what happens if people use the engineering strain to approximate the Maxwell equation. In the *Virlab* system the true strain and true stress can be converted to the engineering strain and the engineering stress. The dash curve is plotted for the relationship between the engineering strain and engineering stress shown in Figure 4.16. This is not an error, but a different view of the same result. The kind of error shown in Figure 2.9 is not possible with the *Virlab* system since we assume that the designer has implemented with the correct specification. The different view of the same result shown in Figure 4.16 may be useful for comparing to experimental data. The dash curve is obviously different with the solid curve. Hence, the correct understanding of the specification of the experiment is very important.

Chapter 5

Conclusions, contributions and future work

In this chapter we first make conclusions for the work presented in this thesis in Section 5.1. We then summarize the contributions of our work in Section 5.2. In Section 5.3 we offers suggestions for future work.

5.1 Conclusions

In this thesis we have provided a framework for material testing and we have described a component-based design. We went through commonality analysis, modularity, component identification, construction of component-based architecture, documentation for MG, MIS and component description and part of the implementation of *Virlab*. Over this experience the following conclusions were made:

- A commonality analysis is necessary for the *Virlab*.

“Commonality analysis is one approach to defining a family by identifying

commonalities, *i.e.*, assumptions that are true for all family members, variabilities, *i.e.*, assumptions about what can vary among family members, and common terminology for the family [AW97].” We started the commonality analysis by analyzing and comparing real material experiments and based on this we assumed unlikely changes for *Virlab* and summarized anticipated changes for *Virlab*. Unlikely changes and anticipated changes are the foundation of the design for *Virlab*.

- Information hiding [Par72] and separation of concern are beneficial for module decomposition principles.

We divided the system into modules by applying information hiding and separation of concern. Based on the anticipated changes from the commonality analysis, we recorded the characteristics of the system that are likely to change and encapsulated each expected change into one module. So each module hides one design decision of the system.

- The use hierarchy shows the use relationship among modules.

“Outside in” [PS75] and “Bottom up” [PS75] approaches are adopted to design the use relation as a hierarchical structure. A layered hierarchical structure of the *Virlab* system makes the responsibilities and functions of the modules clear.

- Modularity is a prerequisite for the component identification and component-based system architecture.

After a good module decomposition, based on the role and function of each module, we migrated parts of the system into components by applying the

principle of separation of concerns from the high level view and divided the system into fundamental components, functional components and application components. The component architecture works as a blueprint for the *Virlab* system development.

- The concept of “Design through documentation” goes through the whole procedure.

In our opinion documentation is both important and useful. It serves as a media for communication between developers, and it can also be used to verify and test the system. So each document is produced at each stage including the module guide, the module interface specification and the component description.

We adapted software engineering approach into the design of *Virlab*. We invested considerable time and effort in each stage and each document was carefully written, checked and reviewed several times. A significant payoff is that the implementation of the *Virlab* system proceeded smoothly. Therefore, we hope that practitioners in the industry can learn some valuable lessons from our experiences.

5.2 Contributions

The contributions of the virtual laboratory for material testing consist of two aspects: one is from the field of material testing and another is from the field of software engineering. Because few people are working on the virtual laboratory for material testing, the contribution for the field of material testing is significant. The contributions are listed below:

- The virtual laboratory for material testing, *Virlab*, supports several classes and types of testing. For example it supports displacement-controlled experiment and load-controlled experiment, and in each type of experiment, it also support a uniaxial experiment, biaxial experiment and multiaxial experiment.
- One of significant contributions is for easy development of new constitutive theories. Because of the complexity of constitutive equations, we delegate the approximation of the constitutive equation to the professionals, *i.e.*, the designer, and *Virlab* supports them adding a new component to the system. When the designers write the new components for the new constitutive theories and link them to the system, they can use *Virlab* to do the experiment under the ideal conditions to test their new theories. *Virlab* deals with the simple cases and ideal conditions so the designers only focus on the development of new constitutive theories. Once they are successful in *Virlab*, they have confidence to do the further test in the commercial simulation package. Therefore, *Virlab* will be helpful for the development of new constitutive equations. Appendix F summarizes the procedures for adding a new component into the *Virlab*.
- The *Virlab* supports the reduction of ambiguity. From the case study discussed in Section 2.3, the clear definitions of strain and stress in the Maxwell equation are important to solve the approximation of the Maxwell equation. If we mistakenly use engineering strain and engineering stress to approximate the Maxwell equation, results are different. The *Virlab* forces the designer and the user to be unambiguous in their definitions; thus, it helps

to avoid errors like that discussed in Section 2.3.

- It can be used for educational purposes. Students can use it to do experiments that designers have built and then they can analyze the experimental data. The students will be able to rapidly investigate many experiments, many materials and the sometimes subtle difference between the various kinematics and stress measures. This will allow students to focus on the mechanics of material, instead of on tedious details.
- It can also be used for research purpose. The *Virlab* offers an convenient platform for researchers to investigate and test new constitutive equation. They can work in the role of designers and provide the implementation of the required *virtual* to allow tests of new material.

The virtual laboratory for material testing has contributions not only for the field of materials testing but also for the field of software engineering. The contributions for the field of software engineering are summarized as follow:

- Providing a framework for the virtual laboratory for material testing. The *Virlab* is an example that helps highlight the challenges of applying software engineering approaches to scientific computing problems in general.
- Demonstrating a systematic approach in developing a virtual laboratory for material testing by software engineering principles and showing an example on how to apply software engineering approaches to practical applications.
- Giving definitions of the terms virtual laboratory and component.
- Conducting a commonality analysis for material testing.

- Providing an easy way to identify components from the module decomposition and then building the component-based system architecture.
- Applying the concept of “design through documentation [PHU81]” in the development of virtual laboratory for material testing and documenting the module guide, module interface specification and component description for *Virllab*.

5.3 Future Work

Although we put considerable effort on researching and designing the virtual laboratory, it is hard for us to make the virtual laboratory for material testing perfect. However, the work on the virtual laboratory for material testing is promising. To make the virtual laboratory for material testing more powerful, there are many avenues for future work. We present some ideas as follows:

- Implementing the remainder of *Virllab*. We only implemented a portion of *Virllab* to act as a proof for concept for our design. Complete implementation will provide more feedback on the design and documentation and will allow others to use the system. These are valuable for further improvement to the system design.
- Building a connection that allows the system to make use of the existing material properties data distributed on World Wide Web in documents using a Hypertext markup language. In the current version of *Virllab*, material properties are specified by input data from the user interface or by reading data from files written by users beforehand. To fully utilize the resource on

the internet, we can adopt the MatML [Web_matml_03] standard to write the programs to obtain the material properties data distributed on the internet and then pass data to *Virlab*. Therefore, a connector between MatML and *Virlab* would be helpful. MatML is an extensible markup language (XML) developed especially for the interchange of material information. It should be possible to build a connector in the future to obtain material properties data by the use of MatML.

- Developing a mathematical parser that allows *Virlab* to handle any mathematical function. We already know that a displacement function in the displacement-controlled experiment, or a load function in the load-controlled experiment, is a function of time. Currently we only support three popular functions, a quadratic function of time, a sin/cosine function of time and an exponential function of time. If a tool for parsing and evaluating general mathematical expressions were integrated into the system it would improve the usefulness of the system.
- Placing the *Virlab* on the World Wide Web to support remote education and research. We adopted COM technologies and component languages to implement the *Virlab*. The widest application of COM technologies is for the distributed system and the features of COM support the Client/Server structure. Most of the current *Virlab* can be migrated to the Server side if new interfaces with the server side are added. The specification component together with new added interfaces with the client side can be changed to work on the client side. Therefore it would be possible to put *Virlab* on the Internet.

- Importing experimental data. In the current version of the *VirLab*, experimental data is created after the completion of the experiment. If outside experimental data from other system or real material experiment can be imported into *VirLab*, this will allow comparison of proposed theories to real data.
- Designing a template for easily adding a new constitutive equation. The goal is to support a network of researcher that can all contribute to a database of materials and material models.
- Adding the ability to do parameter fitting. If real experimental data and a constitutive equation are given, the system will determine the material properties.

Bibliography

- [AJ96] Michael F. Ashby and David R. H. Jones, **Engineering materials I An introduction to their properties and applications 2nd**, Butterworth-Heinemann, 1996
- [AKB+] H. Afsarmanesh, E. C. Kaletas, A. Benabdelkader, C. Garita, L. O. Hertzberger, **Reference Architecture for Scientific Virtual Laboratories**, *Journal of Future Generation Computer Systems*, Vol. 17, No. 8, pp. 999-1008, 2001
- [AW97] M. Ardis and D. Weiss, **Defining Families: The Commonality Analysis**, in *Proceedings of ICSE 19*, pp. 649-650, May 1997
- [BJ81] Ferdinand P. Beer and E. Russel Johnson Jr, **Mechanics of materials** McGRAW-HILL Ryerson Ltd, 1981
- [Boo87] Grady Booch, **Software components with Ada: structures, tools and subsystems**, Addison-Wesley Pub Co, 1991 December
- [Bud01] Muniran Budhu, **Enhancing Instructions Using Interactive Multimedia**, *Simulation*, Vol. 76, No. 4, pp. 222-231, 2001

- [Cle95] Paul C. Clements, **From subroutines to subsystems: component-based software development**, *The American Programmer*, Vol. 8, No. 11, November 1995
- [GAP+02] Manuel Á González, Gloria Arranz, Raúl Portales, Miguel Tamayo and Alberto GonzalezManuel, **Development of a virtual laboratory on the internet as support for physics laboratory training**, *European Journal of Physics*, Vol. 23, pp. 61-67, 2002
- [Hen80] Kathryn L. Heninger, **Specifying software requirement for complex system: New techniques and their application**, *IEEE Transactions on Software Engineering*, Vol. 6, No. 1, pp. 2-13, January 1980
- [HS95] Daniel M. Hoffman and Paul A. Strooper, **Software design, automated, testing, and maintenance: A practical approach**, *International Thomson Computer Press*, 1995
- [HW01] D. M. Hoffman, and D. M. Weiss, editors, **Software Fundamentals - Collected Papers by David L. Parnas**, Addison-Wesley, March 2001
- [Jac93] Ivar Jacobson, **Object-oriented software engineering**, Revised Printing. Addison-Wesley, Reading, MA. 1993
- [JPZ97] R. Janicki, D. L. Parnas, and J. Zucker, **Tabular representations in relational documents**, in *Relational Methods in Computer Science* (C. Brink, W. Kahl, and G. Schmidt, eds.), *Advances in Computing Science*, ch. 12, pp. 184–196, Springer Wien NewYork, 1997.

- [KCZ+01] C. C. Ko, M. Chen, Y. Zhuang and K. C. Tan, **Development of a Web-based Laboratory for control experiments on a coupled tank apparatus**, *IEEE Transaction on Education*, Vol. 44, No. 1, pp. 76-86, February 2001.
- [KJR02] S. K. Khanna, C. H. Jenkins and D. Roylance, **A new approach to integrated instruction in mechanics and material science**, *Proceedings of the institution of mechanical engineers part L-Journal of Materials-Design and Applications*, 216 (L1): 49-53 2002
- [KN96] W. Kozaczynski and Jim Q. Ning, **Component-based software engineering**, *the 4th Conference on Software Reuse*, 1996 (ICSR'96)
- [Mal69] Lawrence E. Malvern, **Introduction to the mechanics of a continuous medium**, Prentice-Hall Inc, 1969
- [Maz70] George E. Maze, **Schaum's outline of theory and problems of continuum mechanics**, McGRAW-HILL Inc, 1970
- [Mer91] L.Mercer. **The Virtual laboratory**. Master's thesis, University of Regina, 1991
- [Par72] David L. Parnas, **On the criteria to be used in decomposing systems into modules**, *Communications of the ACM*, Vol. 15, No. 12, pp. 1053 - 1058, 1972
- [Par74] David L. Parnas, **On a 'buzzword': Hierarchical structure**, *Proc. IFIP Congress '74*, pp.336-339, 1974

- [Par79] David L. Parnas, **Designing software for ease of extension and contraction**, *IEEE Transactions on Software Engineering*, Vol. 5, No. 2, pp. 128-138, March 1979
- [Par84] David L. Parnas, **Some software engineering principles**, *INFOR, Canadian Journal of Operations Research and Information Processing*, Vol. 22, No. 4, pp. 303-316, November 1984.
- [PCW85] David L. Parnas, P. C. Clements, D. M. Weiss, **The modular structure of complex systems**, *Proceedings of the 7th international conference on Software Engineering*, Vol. 11, No. 3, pp. 259-266, March 1985
- [Pet69] Aldor C. Peterson, **Strength of materials**, Allyn and Bacon Inc, 1969
- [PHU81] David L. Parnas, S. Hester, and D. Utter, **Using documentation as a software design medium**, *The Bell System Technical Journal*, Vol. 60, No. 8, pp. 1941-1977, October 1981
- [PS75] D. L. Parnas and D. P. Siewiorek, **Use of the concept of transparency in the design of hierarchically structured systems**, *Communications of the ACM*, Vol. 18, No. 7, pp. 401-408, July 1975
- [PZF00] D. Penumadu, R. Zhao and J. D. Frost, **Virtual geotechnical laboratory experiments using a simulator**, *International Journal of Numerical and Analytical Methods in Geomechanics*, 24, 439-451, 2000
- [REG+00] Irene Luque Ruiz, Enrique López Espinosa, Gonzalo Cerruela García and Miguel Ángel Gómez-Nieto, **Design and development of computer-aided chemical systems: representation and balance**

- of inorganic chemical reactions**, *Journal of chemical information and computer sciences*, Vol. 40, No. 3, pp. 744-752, 2000
- [Rog97] Dale Rogerson, **Inside COM**, Microsoft Press, 1997
- [Sam97] Johannes Sametinger, **Software Engineering with reusable components**, Springer-Verlag, Town. 1997
- [Sch99] Christian Schmid, **A remote laboratory using virtual reality on the web**, *Simulation*, Vol. 73, No. 1, pp. 13-21, 1999
- [Smi93] William F. Smith, **Foundation of materials science and engineering**, McGRAW-HILL Inc, 1993
- [Smi97] Spencer Smith, **Nonisothermal film casting of a viscous fluid**, Master's thesis, McMaster University, 1997
- [Szy99] Clemens Szyperski, **Component Software: Beyond object-oriented programming**, Addison-Wesley/ACM Press, 1999
- [WSS90] Alan Wassyng, Samuel Sharp and Karl Smith, **Personal computers and modeling in engineering education**, *ASEE CoED Journal*, Vol. X, No. 1, pp. 31-46, January-March, 1990
- [Web_mec_02] <http://www.ae.msstate.edu/vlsm>, April, 2002
- [Web_Jhu_02] <http://www.jhu.edu/virtlab.html>, 2002
- [Web_phy_02] <http://physicsweb.org/vlab>
- [Web_Sandia_03] <http://www.sandia.gov/programs/nuclear-weapons/index.html>

[Web_matml_03] <http://www.matml.org>

Appendix A Relationship among kinematics quantities

The purpose of the following table is for easy comparison of the differences and similarities among kinematics quantities. In this table *ud* represents undefined.

Item	Index notation	Expression at current configuration
Material deformation Gradient	$\left[\frac{\partial x_i}{\partial X_j} \right]$	$\begin{bmatrix} (1 + \frac{\bar{u}_1}{L_0}) & ud & ud \\ ud & (1 + \frac{\bar{u}_2}{H_0}) & ud \\ ud & ud & (1 + \frac{\bar{u}_3}{W_0}) \end{bmatrix}$
Spatial deformation gradient	$\left[\frac{\partial X_i}{\partial x_j} \right]$	$\begin{bmatrix} (1 + \frac{\bar{u}_1}{L_0})^{-1} & ud & ud \\ ud & (1 + \frac{\bar{u}_2}{H_0})^{-1} & ud \\ ud & ud & (1 + \frac{\bar{u}_3}{W_0})^{-1} \end{bmatrix}$
Material displacement gradient	$\left[\frac{\partial u_i}{\partial X_j} \right]$	$\begin{bmatrix} \frac{\bar{u}_1}{L_0} & ud & ud \\ ud & \frac{\bar{u}_2}{H_0} & ud \\ ud & ud & \frac{\bar{u}_3}{W_0} \end{bmatrix}$
Spatial displacement gradient	$\left[\frac{\partial u_i}{\partial x_j} \right]$	$\begin{bmatrix} \frac{\bar{u}_1}{L_0 + \bar{u}_1} & ud & ud \\ ud & \frac{\bar{u}_2}{H_0 + \bar{u}_2} & ud \\ ud & ud & \frac{\bar{u}_3}{W_0 + \bar{u}_3} \end{bmatrix}$
Cauchy's deformation tensor	$\left[\frac{\partial X_k}{\partial x_i} \frac{\partial X_k}{\partial x_j} \right]$	$\begin{bmatrix} (1 + \frac{\bar{u}_1}{L_0})^{-2} & ud & ud \\ ud & (1 + \frac{\bar{u}_2}{H_0})^{-2} & ud \\ ud & ud & (1 + \frac{\bar{u}_3}{W_0})^{-2} \end{bmatrix}$

Green's deformation tensor	$\left[\frac{\partial x_k}{\partial X_i} \frac{\partial x_k}{\partial X_j} \right]$	$\begin{bmatrix} (1 + \frac{\bar{u}_1}{L_0})^2 & ud & ud \\ ud & (1 + \frac{\bar{u}_2}{H_0})^2 & ud \\ ud & ud & (1 + \frac{\bar{u}_3}{W_0})^2 \end{bmatrix}$
Lagrangian (Green's) finite strain tensor	$\frac{1}{2} \left(\frac{\partial x_k}{\partial X_i} \frac{\partial x_k}{\partial X_j} - \delta_{ij} \right)$ $\frac{1}{2} \left(\frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i} + \frac{\partial u_k}{\partial X_i} \frac{\partial u_k}{\partial X_j} \right)$	$\begin{bmatrix} \frac{1}{2} \left[(1 + \frac{\bar{u}_1}{L_0})^2 - 1 \right] & ud & ud \\ ud & \frac{1}{2} \left[(1 + \frac{\bar{u}_2}{H_0})^2 - 1 \right] & ud \\ ud & ud & \frac{1}{2} \left[(1 + \frac{\bar{u}_3}{W_0})^2 - 1 \right] \end{bmatrix}$
Eulerian (Almansi's) finite strain tensor	$\frac{1}{2} \left(\frac{\partial x_k}{\partial X_i} \frac{\partial x_k}{\partial X_j} - \delta_{ij} \right)$ $\frac{1}{2} \left(\frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i} + \frac{\partial u_k}{\partial X_i} \frac{\partial u_k}{\partial X_j} \right)$	$\begin{bmatrix} \frac{1}{2} \left[1 - (1 + \frac{\bar{u}_1}{L_0})^{-2} \right] & ud & ud \\ ud & \frac{1}{2} \left[1 - (1 + \frac{\bar{u}_2}{H_0})^{-2} \right] & ud \\ ud & ud & \frac{1}{2} \left[1 - (1 + \frac{\bar{u}_3}{W_0})^{-2} \right] \end{bmatrix}$
Lagrangian infinitesimal strain tensor (Engineering strain tensor)	$\frac{1}{2} \left(\frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i} \right)$	$\begin{bmatrix} \frac{\bar{u}_1}{L_0} & ud & ud \\ ud & \frac{\bar{u}_2}{H_0} & ud \\ ud & ud & \frac{\bar{u}_3}{W_0} \end{bmatrix}$
Eulerian infinitesimal strain tensor	$\frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$	$\begin{bmatrix} \frac{\bar{u}_1}{L_0 + \bar{u}_1} & ud & ud \\ ud & \frac{\bar{u}_2}{H_0 + \bar{u}_2} & ud \\ ud & ud & \frac{\bar{u}_3}{W_0 + \bar{u}_3} \end{bmatrix}$
True strain tensor		$\begin{bmatrix} \ln\left(\frac{L}{L_0}\right) & ud & ud \\ ud & \ln\left(\frac{H}{H_0}\right) & ud \\ ud & ud & \ln\left(\frac{W}{W_0}\right) \end{bmatrix}$

Stretch tensor		$\begin{bmatrix} \Lambda(\hat{e}_1) & ud & ud \\ ud & \Lambda(\hat{e}_2) & ud \\ ud & ud & \Lambda(\hat{e}_3) \end{bmatrix}$
Stretch ratio tensor		$\begin{bmatrix} \frac{1}{\lambda^2(\hat{e}_1)} & ud & ud \\ ud & \frac{1}{\lambda^2(\hat{e}_2)} & ud \\ ud & ud & \frac{1}{\lambda^2(\hat{e}_3)} \end{bmatrix}$

Table A-1 Mathematical relationship among kinematics quantities

Current configurations are the followings:

1. Shear strain is not considered.
2. The experiment is for the multiAxialExperiment shown in Figure A-1
3. Material Physical Features: $L = L_0 + \bar{u}_0, H = H_0 + \bar{u}_2, W = W_0 + \bar{u}_3$
4. x_1, x_2, x_3 is Eulerian Coordinates i.e. Deformed configuration.
5. X_1, X_2, X_3 is Lagrangian Coordinates i.e. Reference configuration
6. $\bar{u}_1, \bar{u}_2, \bar{u}_3$ are displacement at the end of the body in the direction of the coordinate axes
7. u_1, u_2, u_3 as a function of X_1, X_2, X_3
8. L_0, L - Initial and current length, respectively.
 H_0, H - Initial and current height, respectively.
 W_0, W - Initial and current width, respectively.
9. Relationships among $x_1, x_2, x_3, X_1, X_2, X_3, \bar{u}_1, \bar{u}_2, \bar{u}_3$ and u_1, u_2, u_3
 - a. $x_1 = X_1 + u_1, x_2 = X_2 + u_2, x_3 = X_3 + u_3$
 - b. $u_1 = \bar{u}_1 \frac{X_1}{L_0}, u_2 = \bar{u}_2 \frac{X_2}{H_0}, u_3 = \bar{u}_3 \frac{X_3}{W_0}$
 - c. $u_1 = \frac{\bar{u}_1}{L_0 + \bar{u}_1} x_1, u_2 = \frac{\bar{u}_2}{H_0 + \bar{u}_2} x_2, u_3 = \frac{\bar{u}_3}{W_0 + \bar{u}_3} x_3$
 - d. $x_1 = (1 + \frac{\bar{u}_1}{L_0}) X_1, x_2 = (1 + \frac{\bar{u}_2}{H_0}) X_2, x_3 = (1 + \frac{\bar{u}_3}{W_0}) X_3$

e. $X_1 = (1 + \frac{\bar{u}_1}{L_0})^{-1} x_1, X_2 = (1 + \frac{\bar{u}_2}{H_0})^{-1} x_2, X_3 = (1 + \frac{\bar{u}_3}{W_0})^{-1} x_3$

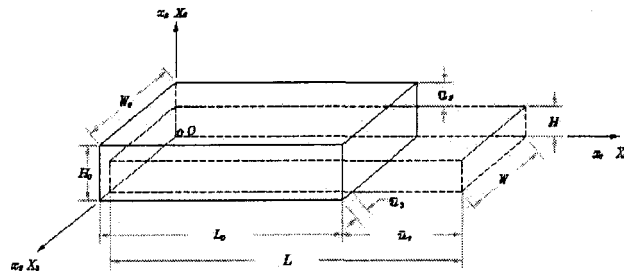


Figure A-1 the multiaxial experiment

Appendix B The solution for the uniaxial extension displacement controlled experiment

This appendix summarizes the closed-form solution to the Maxwell constitutive equation for true strain, true stress and the displacement in the uniaxial extension displacement controlled experiment. The test is shown in Figure B-1.

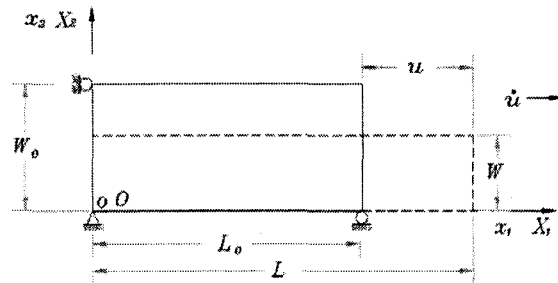


Figure B-1 The uniaxial extension displacement-controlled experiment

The above figure shows the material test by showing the dimensions, boundary conditions and coordinate system. In the figures the test specimen has the initial length (L_0) and width (W_0). The figure also shows that the test specimen is extended by u , which changes at the rate of \dot{u} . In this uniaxial extension displacement controlled experiment assuming that the material is incompressible.

In this experiment, we are given that the rate of extension (velocity) is $\dot{u} = \xi L_0 e^{\xi t}$, ξ is a constant, and we assume a Maxwell equation as the constitutive equation, where the equation can be assumed as $\sigma + \lambda \dot{\sigma} = 2\eta \dot{\epsilon}$. We also know that $L = L_0 + u$ (Please see Section 2.2 in the Chapter 2)

Since the rate of extension \dot{u} is given, the displacement can be determined by integration as follows:

$$u = \int_0^t \dot{u} dt = L_0 (e^{\xi t} - 1) \quad (\text{B.1})$$

and the length (L) as a function of time can be determined using the displacement

$$L = L_0 + L_0(e^{\xi t} - 1) = L_0 e^{\xi t} \quad (\text{B.2})$$

Now the displacement as a function of time is known, so the kinematics quantities can also be summarized as a function of time based on the Appendix A relationship among kinematics quantities.

The discussion can now move to the closed-form solution for stress using the Maxwell equation. In the Maxwell equation used here the correct measure of the strain is the true strain and true strain ε , which is defined as:

$$\varepsilon = \ln \frac{L}{L_0} \quad (\text{B.3})$$

Replace L in the equation B.3 with B.2 and true strain is obtained as follows:

$$\varepsilon = \xi t \quad (\text{B.4})$$

In the Maxwell equation, $\dot{\varepsilon}$ can be determined by the derivation of equation B.4 as follows:

$$\dot{\varepsilon} = \frac{d}{dt}(\xi t) = \xi$$

Now the Maxwell equation can be written as

$$\sigma + \lambda \dot{\sigma} = 2\eta \xi \quad (\text{B.5})$$

So the discussion can move to how to solve this first order linear differential equation.

We know that the solutions for nonhomogenous differential equations are the sum of the homogeneous solution and a particular solution.

At first, to calculate the homogeneous solution of equation B.5, we assume that

$$\sigma + \lambda \dot{\sigma} = 0 \quad (\text{B.6})$$

Assuming

$$\sigma = c_1 e^{-\frac{t}{\lambda}}, c_1 \text{ is taken as a function of time} \quad (\text{B.7})$$

then $\dot{\sigma}$ can be determined by differentiating equation B.7

$$\dot{\sigma} = \dot{c}_1 e^{-\frac{t}{\lambda}} + c_1 e^{-\frac{t}{\lambda}} \left(-\frac{1}{\lambda}\right) \quad (\text{B.8})$$

Second, to calculate the particular solution by substituting B.7 and B.8 into equation B.5, the following results

$$\dot{c}_1 = \frac{2\eta\xi}{\lambda} e^{\frac{t}{\lambda}} \quad (\text{B.9})$$

by differentiation of the equation, the following results

$$c_1 = 2\eta\xi e^{\frac{t}{\lambda}} + c_2 \quad (\text{B.10})$$

so the stress is determined by substituting B.9 and B.10 into B.8

$$\sigma = 2\eta\xi + c_2 e^{-\frac{t}{\lambda}}$$

The final solution can be solved using the initial condition that $\sigma(0) = 0$ to yield

$$\sigma = 2\eta\xi(1 - e^{-\frac{t}{\lambda}}) \quad (\text{B.11})$$

Appendix C The Module Guide for Virlab

Table of Contents

C.1	Overview	129
C.2	Module decomposition.....	130
C.2.1	Behavior hiding modules	130
C.2.2	Hardware hiding modules	130
C.2.3	Software decision hiding modules	130
C.2.4	Use relation	133
C.3	Module guide for behavior hiding modules.....	136
C.3.1	Module guide for behavior hiding modules.....	136
C.3.1.1	Importing constitutive equation module	136
C.3.1.2	Experiment module	136
C.3.1.3	Master control module	136
C.3.1.4	Output show module	137
C.3.1.5	Experiment specification module.....	137
C.3.1.6	Specimen geometry specification module	137
C.3.1.7	Displacement specification module	137
C.3.1.8	Load specification	138
C.3.1.9	Function specification	138
C.3.1.10	Constitutive equation specification module.....	138
C.3.1.11	Material property file module	138
C.3.1.12	Report file module	139
C.3.2	Module guide for software decision hiding modules.....	139
C.3.2.1	Constitutive equation structure module	139
C.3.2.2	Function structure module	139
C.3.2.3	Experiment definition module	140
C.3.2.4	Specimen geometry module.....	140
C.3.2.5	Screen display module	140
C.3.2.6	Table structure module	140
C.3.2.7	Displacement constitutive calculation module	141
C.3.2.8	Load constitutive calculation module	141
C.3.2.9	Material deformation gradient module	141
C.3.2.10	Spatial deformation gradient module.....	142
C.3.2.11	Material displacement gradient module.....	142
C.3.2.12	Spatial displacement gradient module	142
C.3.2.13	Cauchy deformation gradient.....	142
C.3.2.14	Green deformation gradient module	143
C.3.2.15	Stretch tensor module	143
C.3.2.16	Stretch ratio tensor module	143
C.3.2.17	Eulerian infinitesimal strain tensor module	143
C.3.2.18	Lagrangian infinitesimal strain tensor module	144
C.3.2.19	Eulerian finite strain tensor module	144
C.3.2.20	Lagrangian finite strain tensor module	144
C.3.2.21	True strain tensor module	145
C.3.2.22	True stress tensor module	145
C.3.2.23	Engineering stress tensor module	145

- C.3.3 Hardware hiding modules 146
 - C.3.3.1 Keyboard module 146
 - C.3.3.2 Mouse module 146
 - C.3.3.3 Screen display module 146

C.1 Overview

It is difficult to develop a system all at once, especially if it is a large or complex system. A better approach is to divide the task into several modules. “Modules are self-contained systems that can be combined to make large system.” [HW01] Information hiding [DP72] and separation of concerns are the design principles of decomposition that contains three steps:

- Identify the expected changes.
- Encapsulate each expected change. Introduce one module for each change.
- Design the module interface. To each module, an interface will not change if there is a change in the module secret.

A module hides some design decision of the system. There are three classes typically used for the hiding information [PD72]:

- Behavior hiding, such as input formats, screen formats and text messages.
- Software decision hiding, such as algorithms and data structures.
- Machine hiding, such as hardware machine or the “virtual machine”

A module guide gives the secret, service and expected change of each module. Since a module hides a change, the change is called the secret of a module. The module service is the functions that the module provides. The expected change describes the possible change in the future.

The rest of the document is organized into two sections: Section 2 describes the module decomposition and use relation of the system, Section 3 describes the module guide of the system.

C. 2 Module decomposition

This section is organized as follows: Section 2.1 shows the decomposition of the behavior hiding modules, Section 2.2 describes the decomposition of the software decision hiding modules, Section 2.3 gives the decomposition of the hardware hiding modules and Section 2.4 represents the overview of the system with the use relation of the system. In the figures of Section 2 a rectangle with a double frame means this module has some sub modules. Modules shown as rectangles with a single frame are leaf modules, which are the modules that will be implemented.

C. 2.1 Behavior hiding modules

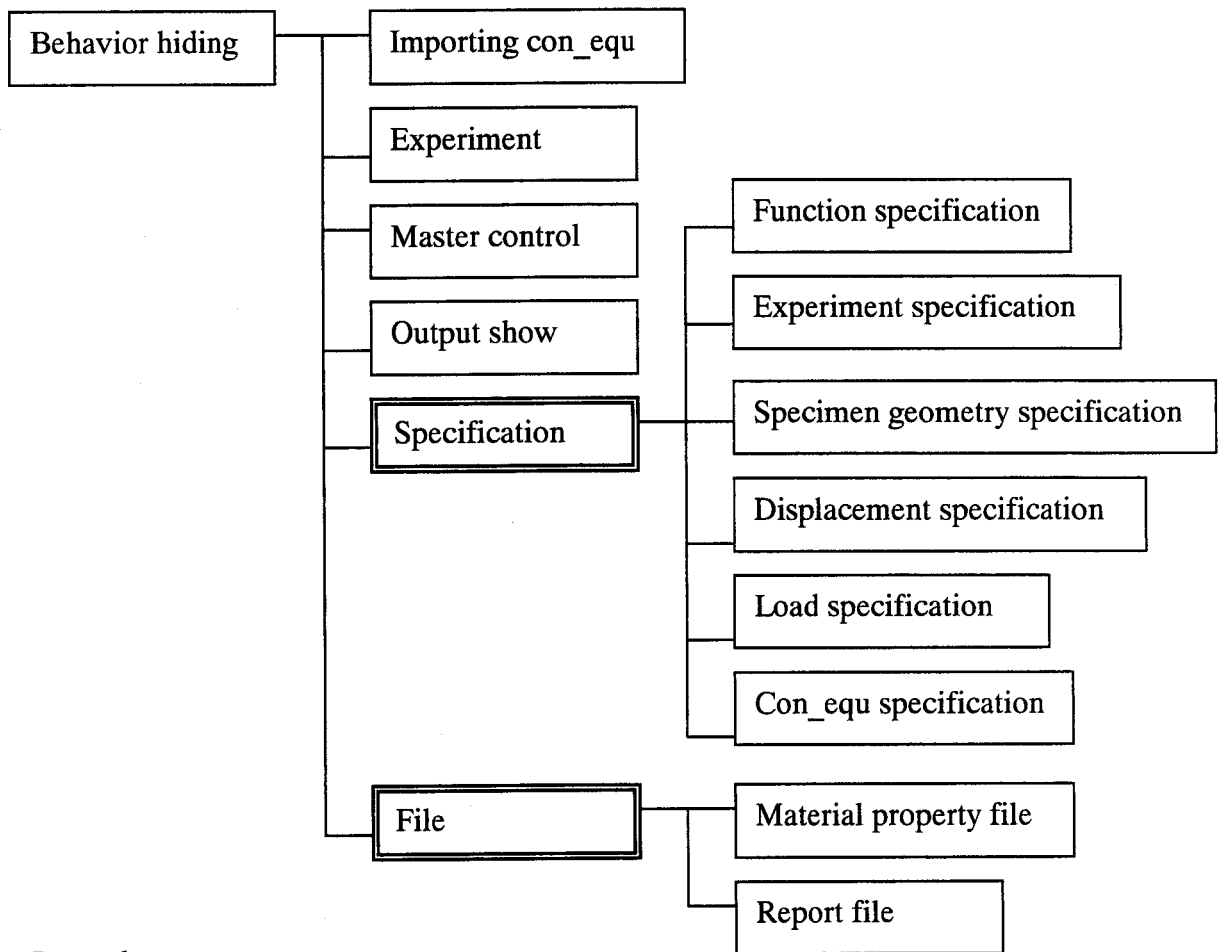
Behavior modules hide the behavior of the system such as input format, screen format and text messages. Figure C-1 shows the system decomposition for the behavior hiding modules.

C. 2.2 Hardware hiding modules

The hardware modules hide the hardware or 'virtual machine' used in the system. The hardware used in this system refers to the keyboard, mouse and screen. We assume that the operating system that the developed system will run on supports keyboard, mouse and screen functions. Figure C-2 shows the decomposition of the machine hiding modules. In this figure modules represented with a dotted frame means that they are provided by the operating system.

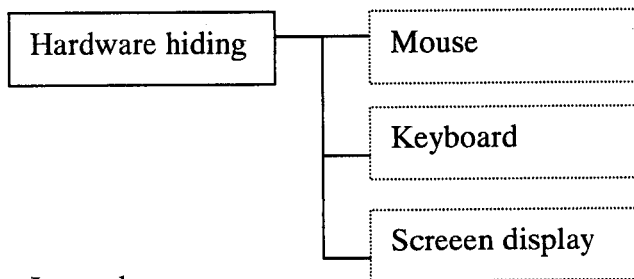
C. 2.3 Software decision hiding modules

Software decision modules hide the system decisions that include data structure and algorithms used in the system. Figure C-3 shows the system decomposition for software decision hiding modules.



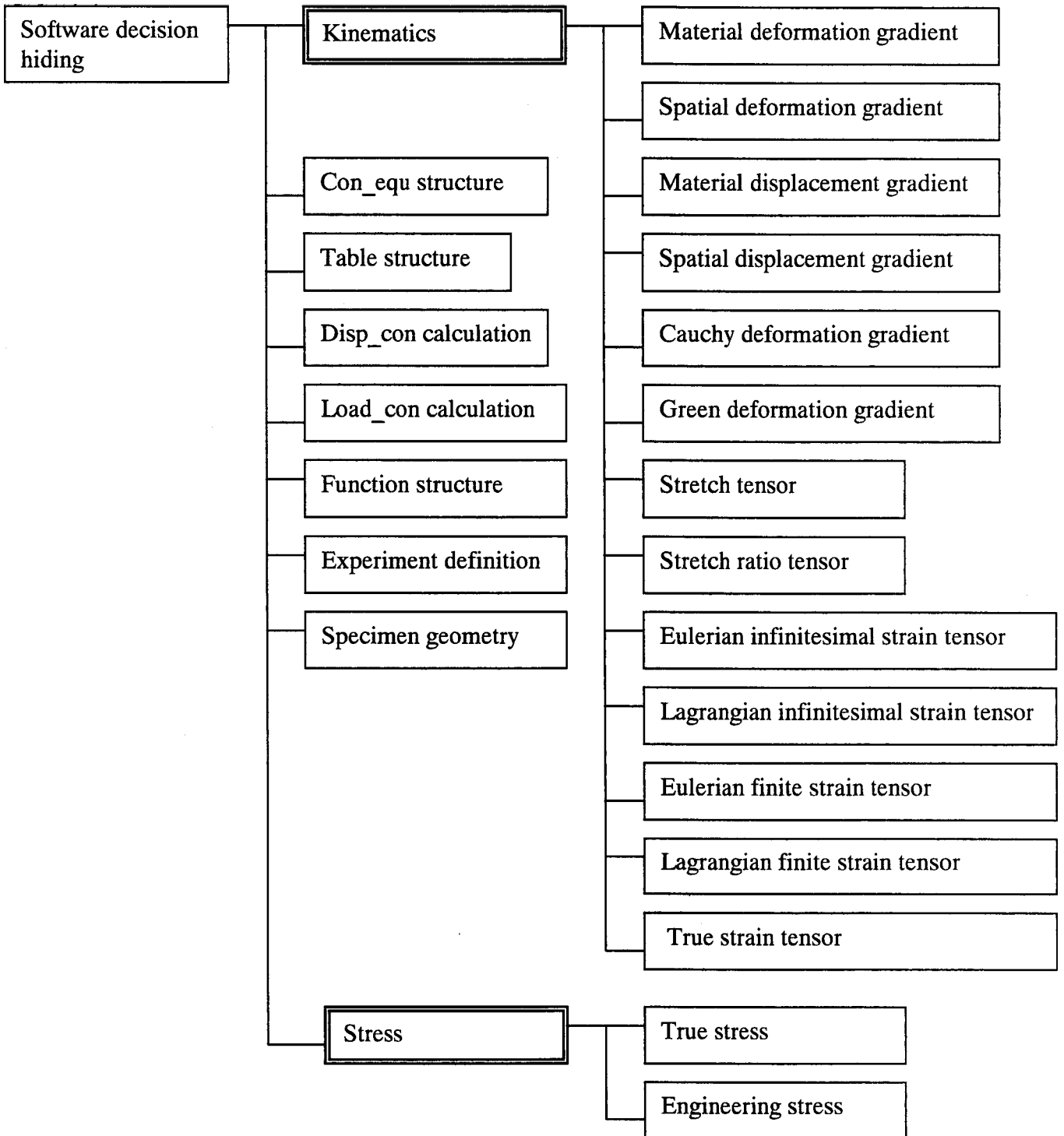
Legend:
 Rectangle with double frame means this module has some sub modules.

Figure C-1 module decomposition of behavior hiding modules



Legend:
 A rectangle with dotted frame means this module is provided by the operating system.

Figure C-2 Module decomposition for the machine hiding modules

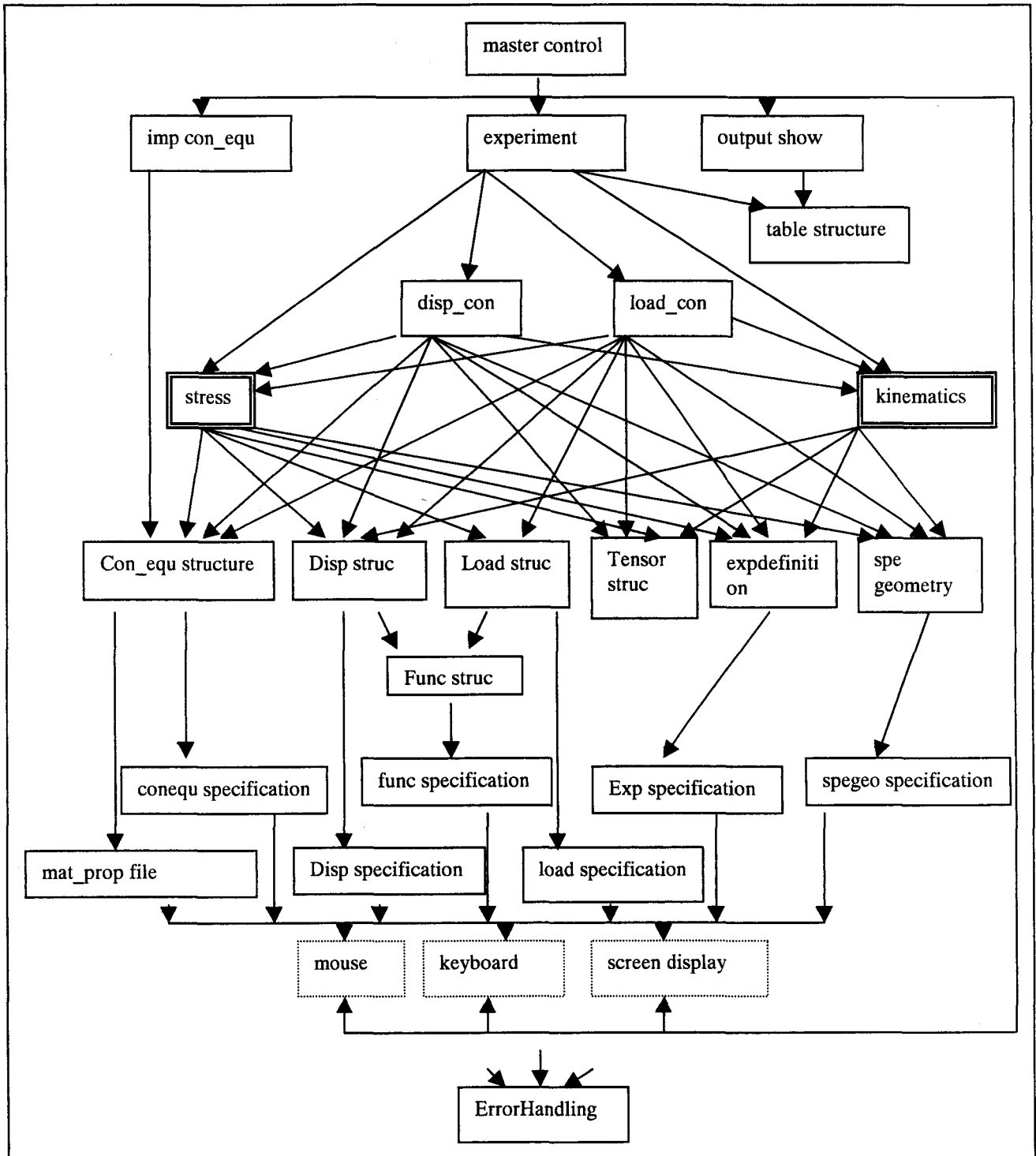


Legend:
 Rectangle with double frame means this module has some sub modules.

Figure C-3 Module decomposition for the software decision hiding modules

C. 2.4 Use relation

Figure C.4 gives the modules that constitute the Virlab system and their uses relation. In this figure, module A uses module B if some programs in the module A rely on a correct implementation of some programs in module B to complete their tasks described in its specification. A rectangle with a single frame means a module and a rectangle with a double frame means a module that has some sub modules and that all the sub modules have the same use relation externally in the use relation of the system. A rectangle with a dotted frame means that a module is assumed to be implemented and provided by the implementation environment. Kinematics Module has some sub modules shown in the Figure C-5. A significant characteristic of the use relation of the kinematics module is that all submodules in the kinematics module can use each other because sometimes known kinematic quantities will be used to calculate an unknown kinematic quantity. All the submodules in the kinematics module have the same use relation externally with the other modules in the system. Following a similar argument, the use relation of the stress module shown in the Figure C-6 has the same characteristics as the use relation of the kinematics module; *i.e.*, all submodules in the stress module can use each other when a known stress is used to calculate an unknown stress and all the submodules in the stress module have the same use relation externally with the other modules in the system.

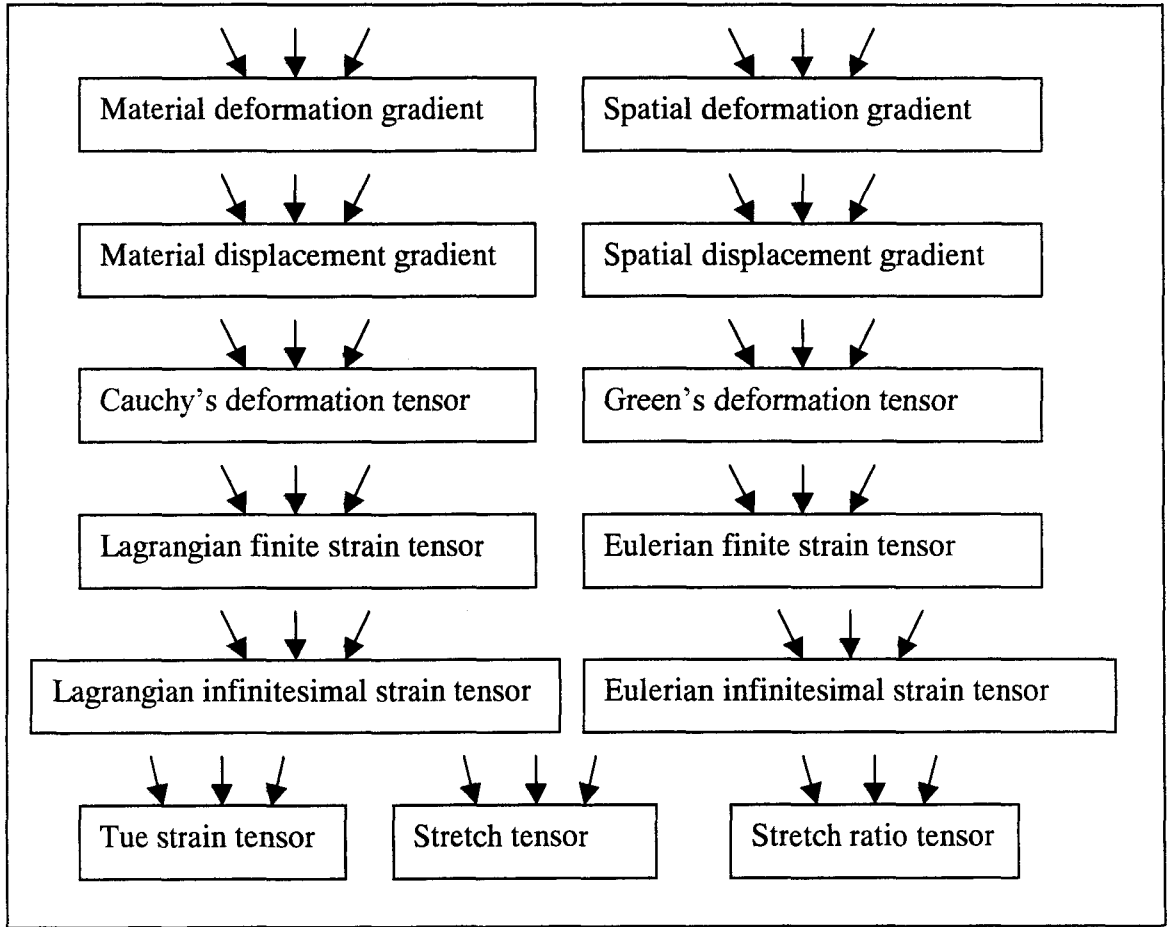


A → B : Module A uses Module B
X : Module X is used by all its upper level modules.

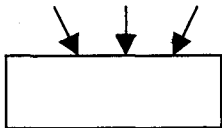
Comment:

disp_con, load_con, con_equ, mat_prop, spe_geo, func and exp are the abbreviations of displacement constitutive, load constitutive, constitutive, material property, geometry, function and experiment respectively.

Figure C.4 Use relation of the system

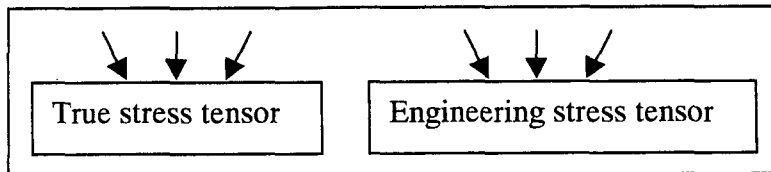


Legend:

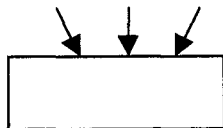


means this module is used by all the other modules.

Figure C-5 use relation in the kinematics module



Legend:



means this module is called by all the other modules.

Figure C-6 Use relation in the stress module

C. 3 Module guide for behavior hiding modules

This section is organized as follows: Section 3.1 gives the module guide for the behavior hiding modules. Section 3.2 provides the module guide for the software decision hiding modules. Section 3.3 shows the module guide for the hardware hiding modules.

C. 3.1 Module guide for behavior hiding modules

This section focuses on the module guide for behavior hiding modules in the *Virlab* system.

C. 3.1.1 Importing constitutive equation module

Module name	Importing constitutive equation module
Module service	Import the new constitutive equation. By this module a new constitutive equation is imported.
Module secret	The calling sequence of modules when a new constitutive equation is imported.
Expected changes	Since this module works as a mediator when a new constitutive equation is imported, the sequence of programs being called might be changed. The expected changes correspond to C_14 in the list of anticipated changes.
Prefix	

C. 3.1.2 Experiment module

Module name	Experiment module
Module service	Finish doing the experiment.
Module secret	The calling sequence of modules to do the experiment.
Expected changes	Since this module works as a mediator when an experiment is done, the sequence of programs being called might be changed. The expected changes correspond to C_14 in the list of anticipated changes.
Prefix	

C. 3.1.3 Master control module

Module name	Master control module
Module service	This module controls the execution sequence of different modules being called through the system.

Module secret	The calling sequence of modules.
Expected changes	Since this module works as a mediator in the whole system, the sequence of programs being called might be changed. The expected changes correspond to C_14 in the list of anticipated changes.
Prefix	

C. 3.1.4 Output show module

Module name	Output show module
Module service	Outputs the experiment data in a curve way or in a table form or in the file and works as an interface between the system and output display.
Module secret	Output format
Expected changes	Output format might be changed or added. The expected changes correspond to C_13 in the list of anticipated changes.
Prefix	os_

C. 3.1.5 Experiment specification module

Module name	Experiment specification module
Module service	Query the user to specify the experiment definition
Module secret	Display content and format.
Expected changes	Display content and format might be changed depending on the requirements. The expected changes correspond to C_11_DEF in the list of anticipated changes.
Prefix	es_

C. 3.1.6 Specimen geometry specification module

Module name	Specimen geometry specification module
Module service	Query the user to specify the geometry of the specimen
Module secret	Display content and format.
Expected changes	Display content and format might be changed if the test specimen is changed. The expected changes correspond to C_11_GEO in the list of anticipated changes.
Prefix	sgs_

C. 3.1.7 Displacement specification module

Module name	Displacement specification module
Module service	Query the user to choose the displacement function
Module secret	Display content and format.
Expected changes	Display content and format might be changed depending on how to describe the displacement. The expected changes correspond to C_11_DISP in the list of anticipated changes.

Prefix	ds
--------	----

C. 3.1.8 Load specification

Module name	Load specification module
Module service	Query the user to choose the load function.
Module secret	Display content and format.
Expected changes	Display content and format might be changed depending on how to describe the load. The expected changes correspond to C 11 LOAD in the list of anticipated changes.
Prefix	ls

C. 3.1.9 Function specification

Module name	Function specification module
Module service	Query the user to specify the function composition
Module secret	Display content and format.
Expected changes	Display content and format might be changed depending on the function format. The expected changes correspond to C_11_FUNC in the list of anticipated changes.
Prefix	fsm

C. 3.1.10 Constitutive equation specification module

Module name	Constitutive equation specification module
Module service	Query the user to specify the constitutive equation
Module secret	Display content and format.
Expected changes	Display content and format might be changed depending on how to describe the constitutive equation. The expected changes correspond to C 11 CON EQU in the list of anticipated changes.
Prefix	ces

C. 3.1.11 Material property file module

Module name	Material property file module
Module service	Provide the material properties of the constitutive equation that are saved into the file and provide the programs to read information on material properties.
Module secret	File format.
Expected changes	The format of file in which material properties are saved might be different. The order of file format is irrelevant but it should be consistent so that the order of information that is read from the file is the same as the order that is written to the file. The expected changes correspond to C 2 in the list of anticipated changes.
Prefix	mpf

C. 3.1.12 Report file module

Module name	Report file module
Module service	Provide the information on experiment configuration and experiment data that are saved in the file and provides programs to write experiment configuration and experiment data into file
Module secret	File format.
Expected changes	File format might be different. The order of file format is irrelevant but it should be consistent so that the order of information that is read from the file is the same as the order that is written to the file. The expected changes correspond to C_13 in the list of anticipated changes.
Prefix	rfm

C. 3.2 Module guide for software decision hiding modules

This section focuses on the module guide for all the software decision hiding modules in the *Virlab* system.

C. 3.2.1 Constitutive equation structure module

Module name	Constitutive equation structure module
Module service	Provide a form to describe the constitutive equation. A data structure is designed to represent this form for the display. This module also provides programs to save information on the constitutive equation to the file and programs to read information on the constitutive equation from the file.
Module secret	Data structure to represent the constitutive equation.
Expected changes	Other data structure might be designed to represent the constitutive equation. A data structure might be adopted to calculate the constitutive equation. The expected changes correspond to C_6 in the list of anticipated changes.
Prefix	cs

C. 3.2.2 Function structure module

Module name	Function structure module
Module service	Provide three functions. Each function is represented with a corresponding data structure and is calculated with an algorithm.
Module secret	Data structure to represent the function and algorithms to calculate the functions.
Expected changes	Other data structure might be designed to represent the function. Or

	new function might be added. The expected changes correspond to C 8 in the list of anticipated changes.
Prefix	fs_

C. 3.2.3 Experiment definition module

Module name	Experiment definition module
Module service	This module provides experiment configuration information.
Module secret	A data structure to represent the experiment configuration.
Expected changes	Since experiment configuration is represented with a data structure, the same experiment might be represented with other data structures depending on designers' choice. The expected changes correspond to C 7 in the list of anticipated changes.
Prefix	ed_

C. 3.2.4 Specimen geometry module

Module name	Specimen geometry module
Module service	This module provides a way to describe the test specimen
Module secret	Geometry of the test specimen is described with a data structure.
Expected changes	If test specimen is changed or geometry of the test specimen is changed, a new data structure might be needed. The expected changes correspond to C 1 in the list of anticipated changes.
Prefix	sg_

C. 3.2.5 Screen display module

Module name	Screen display module
Module service	Displays different components (e.g. buttons, checkbox, icon, etc) that make up the system output interface on the screen. Provides some interfaces between the system and the screen so the system can display information on the screen through the use of the programs in the module.
Module secret	The data structure and the algorithms to display anything on screen. Secret type: software decision hiding
Expected changes	Other data structure might be designed to display user interface elements on screen.
Prefix	

C. 3.2.6 Table structure module

Module name	Table structure module
Module service	This module provides a data structure to save all the data and also provides programs to save data into the table and to read experiment data from the table.

Module secret	The data structure and the algorithms to save the experiment data.
Expected changes	Other data structure might be designed to save the data. The expected changes correspond to C_12 in the list of anticipated changes.
Prefix	ts

C. 3.2.7 Displacement constitutive calculation module

Module name	Displacement constitutive calculation module
Module service	Provide the numerical approximation of the constitutive equation to obtain the stress or other kinematics quantities when the constitutive equation is introduced in the displacement experiment.
Module secret	Algorithm to calculate the numerical approximation of the constitutive equation in the displacement controlled experiment.
Expected changes	Different algorithm might be adopted to calculate the approximation. The expected changes correspond to C_3_DISP in the list of anticipated changes.
Prefix	dcc

C. 3.2.8 Load constitutive calculation module

Module name	Load constitutive calculation module
Module service	Provide the numerical approximation of the constitutive equation to obtain the displacement and then other kinematics quantities when the constitutive equation is introduced in the load experiment.
Module secret	Algorithm to calculate the numerical approximation of the constitutive equation in the load controlled experiment.
Expected changes	Different algorithm might be adopted to calculate the approximation. The expected changes correspond to C_3_LOAD in the list of anticipated changes.
Prefix	lcc

C. 3.2.9 Material deformation gradient module

Module name	Material deformation gradient module
Module service	Provides the programs to obtain the material deformation gradient
Module secret	The calculating procedure to obtain the material deformation gradient.
Expected changes	F is called the material deformation gradient by George E Mase(1970). Different researcher might give it a different name. User can get the material deformation gradient based on the different known conditions. The expected changes correspond to C 4 MDG in the list of anticipated changes.
Prefix	mdg

C. 3.2.10 Spatial deformation gradient module

Module name	Spatial deformation gradient module
Module service	Provides the programs to obtain the spatial deformation gradient
Module secret	The calculating procedure to obtain the spatial deformation gradient.
Expected changes	H is called the spatial deformation gradient by George E. Mase(1970). Different researcher might give it a different name and also use different algorithms to get the spatial deformation gradient. The expected changes correspond to C_4_SDG in the list of anticipated changes.
Prefix	sdg_

C. 3.2.11 Material displacement gradient module

Module name	Material displacement gradient module
Module service	Provides the programs to obtain the material displacement gradient
Module secret	The calculating procedure to obtain the material displacement gradient. Secret type: software decision hiding
Expected changes	J is called the material displacement gradient by George E. Mase(1970). Different researcher might give different name for the same tensor. Different researchers can change algorithm to get the material displacement gradient. The expected changes correspond to C 4 MDPG in the list of anticipated changes.
Prefix	mdpg_

C. 3.2.12 Spatial displacement gradient module

Module name	Spatial displacement gradient module
Module service	Provides the programs to obtain the spatial displacement gradient
Module secret	The calculating procedure to obtain the spatial displacement gradient. Secret type: software decision hiding
Expected changes	K is called the spatial displacement gradient by George E. Mase(1970). Different researcher might give it a different name. Different researchers can change algorithm to get the spatial displacement gradient. The expected changes correspond to C 4 SDPG in the list of anticipated changes.
Prefix	sdpg_

C. 3.2.13 Cauchy deformation gradient

Module name	Cauchy deformation gradient module
Module service	Provides the programs to obtain the Cauchy deformation gradient
Module secret	The calculating procedure to obtain the Cauchy deformation gradient. Secret type: software decision hiding
Expected changes	C is called the Cauchy's deformation tensor by George E. Mase

	(1970). Different researcher might give it a different name and use different algorithms to calculate the Cauchy deformation tensor. The expected changes correspond to C_4_CDG in the list of anticipated changes.
Prefix	cdt

C. 3.2.14 Green deformation gradient module

Module name	Green deformation gradient module
Module service	Provides the programs to obtain the Green deformation gradient
Module secret	The calculating procedure to obtain the Green deformation gradient. Secret type: software decision hiding
Expected changes	G is called the Green deformation tensor by George E. Mase (1970). Different researcher might give it a different name and use different algorithms to calculate the Green deformation gradient. The expected changes correspond to C_4_GDG in the list of anticipated changes.
Prefix	gdt

C. 3.2.15 Stretch tensor module

Module name	Stretch tensor
Module service	Provides the programs to obtain the stretch tensor
Module secret	The calculating procedure to obtain the stretch tensor. Secret type: software decision hiding
Expected changes	The stretch tensor is named by George E. Mase (1970). Different researcher might give it a different name. The expected changes correspond to C_4_ST in the list of anticipated changes.
Prefix	st

C. 3.2.16 Stretch ratio tensor module

Module name	Stretch ratio tensor
Module service	Provides the programs to obtain the stretch ratio tensor
Module secret	The calculating procedure to obtain the stretch ratio tensor. Secret type: software decision hiding
Expected changes	The stretch tensor is named by George E. Mase (1970). Different researcher might give it a different name. The expected changes correspond to C_4_SRT in the list of anticipated changes.
Prefix	srt

C. 3.2.17 Eulerian infinitesimal strain tensor module

Module name	Eulerian infinitesimal strain tensor module
-------------	---

Module service	Provides the programs to obtain the Eulerian infinitesimal strain tensor.
Module secret	The calculating procedure to obtain the Eulerian infinitesimal strain tensor. Secret type: software decision hiding
Expected changes	e is called the Eulerian infinitesimal strain tensor by George E. Mase (1970). Different researchers might give it a different name and they also might use other algorithms to calculate the Eulerian infinitesimal strain tensor. The expected changes correspond to C_4_EIST in the list of anticipated changes.
Prefix	eist

C. 3.2.18 Lagrangian infinitesimal strain tensor module

Module name	Lagrangian infinitesimal strain tensor module
Module service	Provides the programs to obtain the Lagrangian infinitesimal strain tensor.
Module secret	The calculating procedure to obtain the Lagrangian infinitesimal strain tensor. Secret type: software decision hiding
Expected changes	l is called the Lagrangian infinitesimal strain tensor named by George E. Mase (1970). Different researchers might give it a different name and they might adopt different algorithms to calculate the Lagrangian infinitesimal strain tensor. The expected changes correspond to C_4_LIST in the list of anticipated changes. Lagrangian infinitesimal strain tensor is also called engineering strain tensor.
Prefix	list

C. 3.2.19 Eulerian finite strain tensor module

Module name	Eulerian finite strain tensor module
Module service	Provides the programs to obtain the Eulerian finite strain tensor.
Module secret	The calculating procedure to obtain the Eulerian finite strain tensor. Secret type: software decision hiding
Expected changes	E is called the Eulerian (Almansi's) finite strain tensor by George E. Mase (1970). Different researcher might give it a different name and they also might use other algorithms to calculate the Eulerian finite strain tensor. The expected changes correspond to C_4_EFST in the list of anticipated changes.
Prefix	efst

C. 3.2.20 Lagrangian finite strain tensor module

Module name	Lagrangian finite strain tensor module
Module service	Provides the programs to obtain the Lagrangian finite strain tensor.
Module secret	The calculating procedure to obtain the Lagrangian finite strain tensor. Secret type: software decision hiding

Expected changes	L is called the Lagrangian (Green) finite strain tensor by George E. Mase (1970). Different researcher might give it a different name and also they might use other algorithms to calculate the Lagrangian finite strain tensor. The expected changes correspond to C 4 LFST in the list of anticipated changes.
Prefix	lfst

C. 3.2.21 True strain tensor module

Module name	True strain tensor module
Module service	Gives the definition of the true strain and provides the programs to obtain the true strain tensor.
Module secret	The calculating procedure to obtain the true finite strain tensor. Secret type: software decision hiding
Expected changes	Different researcher might give it a different name. The expected changes correspond to C 4 TST in the list of anticipated changes.
Prefix	tst

C. 3.2.22 True stress tensor module

Module name	True stress tensor module
Module service	Gives the definition of the true stress. To obtain the true stress tensor, constitutive equation is required. The numerical approximation of the constitutive equation can be calculated. Provides the programs to obtain the true stress tensor.
Module secret	Algorithms to obtain the numerical approximation of the constitutive equation. Secret type: software decision hiding
Expected changes	Different algorithms might be adopted depending on the designer's choice. The expected changes correspond to C_5_TSST in the list of anticipated changes.
Prefix	tsst

C. 3.2.23 Engineering stress tensor module

Module name	Engineering stress tensor module
Module service	Gives the definition of the engineering stress and Provides the programs to obtain the Engineering stress tensor.
Module secret	The calculating procedure to obtain the engineering stress tensor. Secret type: software decision hiding
Expected changes	Different researcher might give it a different name. The expected changes correspond to C 5 ESST in the list of anticipated changes.
Prefix	esst

C. 3.3 Hardware hiding modules

This section provides the module guide for all the hardware hiding modules in the *Virlab* system.

C. 3.3.1 Keyboard module

Module name	Keyboard module
Module service	Keyboard works as a bridge between system software and user. This module provides all keyboard events that system software needs to respond.
Module secret	Keyboard event. Secret type: Hardware-hiding
Expected changes	Other keyboard event might be added to the system.
Prefix	

C. 3.3.2 Mouse module

Module name	Mouse module
Module service	Mouse also works as a bridge between system software and user. This module provides all mouse events that system software needs to respond.
Module secret	Mouse event. Secret type: Hardware-hiding
Expected changes	Other mouse event might be added to the system based on the need.
Prefix	

C. 3.3.3 Screen display module

Module name	Screen display module
Module service	This module provides screen display functions that system software needs to respond.
Module secret	Screen information. Secret type: Hardware-hiding
Expected changes	Other screen display functions might be added to the system based on the need.
Prefix	

Appendix D **The Module Interface Specification for *Virlab***

Table of Contents

D. 1	Introduction.....	150
D. 2	Convention.....	150
D. 3	Module Interface Specification.....	153
D. 3.1	Experiment definition module	153
D. 3.1.1	Interface syntax.....	153
D. 3.1.2	Interface semantics.....	154
D. 3.2	Specimen geometry module.....	157
D. 3.2.1	Interface Syntax	157
D. 3.2.2	Interface semantics.....	157
D. 3.3	Function structure module	159
D. 3.3.1	Interface syntax.....	159
D. 3.3.2	Interface semantics.....	160
D. 3.4	Displacement specification module	162
D. 3.4.1	Interface syntax.....	162
D. 3.4.2	Interface semantics.....	163
D. 3.5	Load specification module.....	165
D. 3.5.1	Interface syntax.....	165
D. 3.5.2	Interface semantics.....	166
D. 3.6	Table structure module	168
D. 3.6.1	Interface syntax.....	168
D. 3.6.2	Interface semantics.....	168
D. 3.7	Screen display module	171
D. 3.8	Experiment specification module.....	172
D. 3.8.1	Interface syntax	172
D. 3.8.2	Interface semantics.....	172
D. 3.9	Function specification module	174
D. 3.9.1	Interface syntax	174
D. 3.9.2	Interface semantics.....	174
D. 3.10	Specimen geometry specification module	176
D. 3.10.1	Interface syntax	176
D. 3.10.2	Interface semantics.....	176
D. 3.11	Material properties file module.....	178
D. 3.11.1	Interface syntax	178
D. 3.11.2	Interface semantics.....	178
D. 3.12	Constitutive equation structure module	180
D. 3.12.1	Interface syntax	180
D. 3.12.2	Interface semantics.....	181
D. 3.13	Constitutive equation specification module.....	183
D. 3.13.1	Interface syntax	183
D. 3.13.2	Interface semantics.....	183
D. 3.14	Tensor data definition module	185

D. 3.14.1	Interface syntax	185
D. 3.15	Material deformation gradient module	187
D. 3.15.1	Interface syntax	187
D. 3.15.2	Interface semantics.....	187
D. 3.16	Spatial deformation gradient module	192
D. 3.16.1	Interface syntax	192
D. 3.16.2	Interface semantics.....	192
D. 3.17	Material displacement gradient module	197
D. 3.17.1	Interface syntax	197
D. 3.17.2	Interface semantics.....	197
D. 3.18	Spatial displacement gradient module	202
D. 3.18.1	Interface syntax	202
D. 3.18.2	Interface semantics.....	202
D. 3.19	Cauchy deformation tensor module	207
D. 3.19.1	Interface syntax	207
D. 3.19.2	Interface semantics.....	207
D. 3.20	Green deformation tensor module.....	212
D. 3.20.1	Interface syntax	212
D. 3.20.2	Interface semantics.....	212
D. 3.21	Lagrangian (Green's) finite strain tensor module	217
D. 3.21.1	Interface syntax	217
D. 3.21.2	Interface semantics.....	217
D. 3.22	Eulerian (Almansi's) finite strain tensor module	223
D. 3.22.1	Interface syntax	223
D. 3.22.2	Interface semantics.....	223
D. 3.23	Lagrangian (Green's) infinitesimal strain tensor module	229
D. 3.23.1	Interface syntax	229
D. 3.23.2	Interface semantics.....	229
D. 3.24	Eulerian (Almansi's) infinitesimal strain tensor module	234
D. 3.24.1	Interface syntax	234
D. 3.24.2	Interface semantics.....	234
D. 3.25	True strain tensor module	239
D. 3.25.1	Interface syntax	239
D. 3.25.2	Interface semantics.....	239
D. 3.26	Displacement constitutive calculation module	245
D. 3.26.1	Interface syntax	245
D. 3.26.2	Interface semantics.....	245
D. 3.27	Load constitutive calculation module	247
D. 3.27.1	Interface syntax	247
D. 3.27.2	Interface semantics.....	247
D. 3.28	Engineering stress module	248
D. 3.28.1	Interface syntax	248
D. 3.28.2	Interface semantic	248
D. 3.29	True stress module	251
D. 3.29.1	Interface syntax	251
D. 3.29.2	Interface semantics.....	251

D. 3.30	Output show module	253
D. 3.30.1	Interface syntax	253
D. 3.30.2	Interface semantics.....	253

D. 1 Introduction

A module interface specification describes the detailed design of the system. It is represented as a collection of access routines. There are common idiom that help with the design of a set of access routines. The idoms are available for set, sequence and tuple types. This appendix is known as the module interface specification (MIS) for the *Virlab* software.

This appendix is organized as follows: Section 2 includes some of the conventions we adapted in documenting the MIS with the attempt to remove ambiguity for readers. Sections 3 shows the module interface specifications for the modules, each of which comprises a syntax subsection and a semantics subsection.

D. 2 Convention

- Naming convention
 - Generally the access routine name includes three parts: the first is the prefix that is the same as the module's prefix, so all the access routines with the same prefix mean that they serve for the same module. The second is a letter s and/or g that explain that the second part is only included for set and/or get access programs. The third is the routine name. The three parts are separated by ' _ '.
 - Some access routine names are just composed of prefix and routine name because this access routine does not involve with the state.

- That some exported functions are named without the prefix means that they are more related with the development environment and are assumed to be provided by the development environment or operating systems.
- The letters in the names of constants are capitalized.
- Mathematics definition
 - ‘:=’ means assignment
 - ‘=’ means comparison
 - ‘ \Rightarrow ’ means a condition rule.
 - ‘ \in ’ and ‘ \notin ’ follow their meanings in discrete mathematics
- Type definition
 - The primary types are string, real, boolean and integer.
 - Real^* is a new type and is equal to a set of all real numbers plus a special value named *undefined*, that is, $\text{Real}^* = \mathbb{R} \cup \{\text{undefined}\}$. Many of the data structures that are introduced use a sequence of Real^* , *ud* is the abbreviation of the special value ‘*undefined*’.
 - Set, sequence and tuple are used in the type constructor.
 - User-defined types in the module interface specification start with a capital letter.
 - ‘*a: Type*’ means *a* is of type ‘*Type*’.
- Developer/designer/user
 - User is a person who will do the experiment with VirLab software; For example, a user might be a student.

- Designer is a person who designs an experiment with Virlab software for a user to do the experiment or for himself to do the research. A designer is a professional with the professional knowledge that a material experiment needs.
- Developer is a person who designs the Virlab software. A developer should have software engineering knowledge.
- ‘Virtual’ in the module interface specification
 - The term ‘virtual’ is used in object-oriented programming language and means if the method is defined as virtual in the base class, its concrete implementation will be provided in the subclass of this base class. Here the term ‘virtual’ is borrowed to define the attribute of the access programs in the module interface specification. It represents that the developers have responsibilities to provide the implementation of the corresponding access programs.
- @ in the event table

The notation @Click(buttonname) is used to denote the occurrence of the event that button ‘buttonname’ is clicked. For example, the event @Click(Confirm) occurs when the ‘Confirm’ button is clicked. Other events include @Click(Cancel), @Click(Save) etc.

D. 3 Module Interface Specification

This section includes the module interface specification (MIS) for the modules designed for the *VirLab* software. Each of MIS is composed of a syntax subsection and a semantics subsection.

D. 3.1 Experiment definition module

Prefix: ed_

Reference: MG – C. 3.2.3

D. 3.1.1 Interface syntax

Exported data types:

ExperimentClassT = set of {displacement-controlled, load-controlled}

ExperimentTypeT = set of {uniaxial, biaxial, multiaxial}

CoordinateTypeT = set of {superimposed, nonsuperimposed}

FunctionTypeT = {quafunction, expfunction, cosfunction}

FunctionDefinitionT = set of FunctionTypeT

ConEquSwitchT = set of {consider_con_equ, nonconsider_con_equ}

TimeDefinitionT = tuple of {inittime: real; timelength: real; timestep: real}

CoordinateDataT = tuple of {angle: real, setoff: real}

Exported constants: none

Exported functions:

Routines name	Inputs	Outputs	Exception
ed_s_experimentclass	ExperimentClassT		none
ed_g_experimentclass		ExperimentClassT	none
ed_s_experimenttype	ExperimentTypeT		none
ed_g_experimenttype		ExperimentTypeT	none
ed_s_coordinatetype	CoordinateTypeT		none
ed_g_coordinatetype		CoordinateTypeT	none
ed_s_functiondefinition	FunctionDefinitionT		none
ed_g_functiondefinition		FunctionDefinitionT	none
ed_s_conequswitch	ConEquSwitchT		none
ed_g_conequswitch		ConEquSwitchT	none
ed_s_timedefinition	TimeDefinitionT		none
ed_g_timedefinition		TimeDefinitionT	none
ed_s_coordinatedata	CoordinateDataT		none
ed_g_coordinatedata		CoordinateDataT	none

D. 3.1.2 Interface semantics

State variables:

expClass: ExperimentClassT
expType: ExperimentTypeT
coord: CoordinateTypeT
func: FunctionDefintionT
time: TimeDefinitionT
con_switch: ConEquSwitchT
coorddata: CoordinateDataT

State invariants: none

Assumptions:

- State coord is set to superimposed by default. In this version the case that state coordinate is equal to nonsuperimposed is not considered. If state coord is equal to nonsuperimposed, type constructor CoordinateDataT will be used and state coorddata will be set. So type CoordinateDataT and exported functions ed_s_CoordinateData and ed_g_CoordinateData are designed for future expansion.

Access routines semantics:

ed_s_experimentclass(c: ExperimentClassT)
exception none
transition expClass := c

ed_g_experimentclass
exception: none
output out := expClass

ed_s_experimenttype(t: ExperimentTypeT)
exception none
transition expType := t

ed_g_experimenttype
exception: none
output out := expType

ed_s_coordinatetype(c: CoordinateTypeT)
exception none
transition coord := c

ed_g_coordinatetype
exception: none
output out := coord

ed_s_functiontype(f: FunctionTypeT)

exception none
transition func := f

ed_g_functiontype

exception: none
output out := func

ed_s_conequswitch(s: ConEquSwitcT)

exception none
transition con_switch := s

ed_g_conequswitch

exception: none
output out := con_switch

ed_s_timedefintion(t: TimeDefinitionT)

exception none
transition time := t

ed_g_timedefinition

exception: none
output out := time

Comments:

1. quafunction, expfunction and cosfunction

The function type refers to the type of displacement function or load function used in the experiment. Three popular mathematical function types are chosen: quadratic function, sine/cosine function, exponential function. Table 4.4.1-1 gives the concrete type of each function.

quafunction	expfunction	cosfunction
$ax^2 + bx + c$	$c_1 e^{\alpha t}$	$c_1 \sin(\alpha t) + c_2 \cos(\beta t)$

Table D-1 Function type

2. displacement-controlled and load-controlled

In the definition of ExperimentClassT displacement-controlled refers to the displacement controlled experiment and load-controlled refers to the load controlled experiment.

3. superimposed and nonsuperimposed

These two terms are used in the definition of CoordinateTypeT. Superimposed means that Lagrangian coordinate system and Eulerian coordinate system mentioned at section 2.1 in chapter two have the same origin and further their axes are parallel. Nonsuperimposed means that Lagrangian coordinate system and

Eulerian coordinate system do not have the same origin or their axes are not parallel in both cases.

4. `consider_con_equ` and `nonconsider_con_equ`

In the definition of `ConEquSwitchT`, `consider_con_equ` refers to the case when the constitutive equation is introduced in the experiment and also is used to obtain the experimental data such as kinematics quantities or stress. `nonconsider_con_equ` refers to the case that the experimental data of interest can be calculated without introducing a constitutive equation.

D. 3.2 Specimen geometry module

Prefix: sg_

Reference: MG – C. 3.2.4

D. 3.2.1 Interface Syntax

Exported data types:

```
SpecimenGeometryT = tuple of {  
    length: real  
    height: real  
    width: real  
    anglexy: real /* design for the future*/  
    angleyz: real  
    anglexz: real  
}
```

Exported constants: none

Exported functions:

Routines name	Inputs	Outputs	Exception
sg_s_geometry	real, real, real, real, real, real		none
sg_g_geometrylength		real	none
sg_g_geometryheight		real	none
sg_g_geometrywidth		real	none
sg_g_geometryanglexy		real	none
sg_g_geometryangleyz		real	none
sg_g_geometryanglexz		real	none

D. 3.2.2 Interface semantics

State variables:

spe: SpecimenGeometryT

State invariants: none

Assumptions:

- Since the test specimen is assumed as a rectangle box, anglexy, angleyz and anglexz are assumed as ninety degrees. Exported functions sg_g_geometryanglexy, sg_g_geometryangleyz and sg_g_geometryanglexz are designed for the future.

Access routines semantics

sg_s_geometry(l: real, h: real, w: real, axy: real, ayz: real, axz: real)
exception None

transition spe.length := l
 spe.height := h
 spe.width := w
 spe.anglexy := axy
 spe.angleyz := ayz
 spe.anglexz := axz

ed_g_geometrylength()

exception: none
output: out := spe.length

ed_g_geometryheight()

exception: none
output: out := spe.height

ed_g_geometrywidth()

exception: none
output: out := spe.width

ed_g_geometryanglexy()

exception: none
output: out := spe.anglexy

ed_g_geometryangleyz()

exception: none
output out := spe.angleyz

ed_g_geometryanglexz()

exception: none
output: out := spe.anglexz

Comments:

1. anglexy in the SpecimenGeometryT refers to the angle between the length side and height side of the rectangle box, angleyz in the SpecimenGeometryT refers to the angle between the length side and width side of the rectangle box, anglexz in the SpecimenGeometryT refers to the angle between the height side and width side of the rectangle box.

D. 3.3 Function structure module

Prefix: fs_

Reference: MG – C. 3.2.2

D. 3.3.1 Interface syntax

Imported data types:

- FunctionDefinitionT from experiment definition module
FunctionDefinitionT = set of FunctionTypeT
FunctionTypeT = {quafunction, expfunction, cosfunction}

External functions:

Sinfunction(t:real)

Cosfunction(t:real)

Expfunction(t: real)

ed_g_functiondefinition from the experiment definition module

Exported data types:

QuaDefinitionT = tuple of {
qua_coefficient: real
lin_coefficient: real
con_coefficient: real

}/* $f(t) = qua_coefficient * t^2 + lin_coefficient * t + con_coefficient$ */

CosDefinitionT = tuple of {
sin_coefficient: real
sin_t_coefficient: real
cos_coefficient: real
cos_t_coefficient: real

}/* $f(t) = sin_coefficient * sin(sin_t_coefficient * t) +$
 $cos_coefficient * cos(cos_t_coefficient * t)$ */

ExpDefinitionT = tuple of {
exp_coefficient: real
exp_t_coefficient: real

} /* $f(t) = exp_coefficient * e^{exp_t_coefficient * t}$ */

Exported constants: none

Exported functions:

Routines name	Inputs	Outputs	Exception
fs_s_initqua			
fs_s_initcos			
fs_s_initexp			
fs_s_quafunction	real, real, real		
fs_s_expfunction	real, real		
fs_s_cosfunction	real, real, real, real		

fs_g_quafunctionvalue	t: real	real	
fs_g_expfunctionvalue	t: real	real	
fs_g_cosfunctionvalue	t: real	real	
fs_g_functionvalue	ft: FunctionTypeT, t: real	real	

D. 3.3.2 Interface semantics

State variables

qua_state: QuaDefintionT
 cos_state: CosDefintionT
 exp_state: ExpDefintionT
 qua_set: boolean := FALSE
 cos_set: boolean := FALSE
 exp_set: boolean := FALSE

State invariants: none

Assumptions:

- Experiment definition module should be initialized before function structure module is called.

Access routines semantics

fs_s_quafunction(q: real, l: real, c: real)

exception: None
 transition: qua_set := TRUE
 qua_state.qua_coefficient := q, qua_state.lin_coefficient := l,
 qua_state.qua_con_coefficient:= c

fs_s_cosfunction(c: real, ct: real, s: real, st: real)

exception: None
 transition: cos_set := TRUE
 cos_state.cos_coefficient := c,
 qua_state.cos_t_coefficient := ct,
 qua_state.sin_coefficient:= s,
 qua_state.sin_t_coefficient:=st

fs_s_expfunction(e: real, et: real)

exception: none
 transition: exp_set := TRUE
 exp_state.exp_coefficient := e,
 exp_state.exp_t_coefficient := et

fs_g_quafunctionvalue(t: real)

exception: (qua_set = FALSE) ⇒ NotSet

output: out := qua_state.qua_coefficient*t*t +
 qua_state.lin_coefficient*t+
 qua_state.con_coefficient

fs_g_cosfunctionvalue(t: real)

exception: (cos_set = FALSE) ⇒ NotSet

output: out :=
 cos_state.cos_coefficient*Cosfunction(cos_state.cos_t_coefficient*t)
 +cos_state.sin_coefficient*Sinfuction(sin_state.sin_t_coefficient*t)

fs_g_expfunctionvalue(t: real)

exception: (exp_set = FALSE) ⇒ NotSet

output: out :=
 exp_state.exp_coefficient*Expfunction(exp_state.exp_t_coefficient*t)

fs_g_functionvalue(ft: FunctionTypeT, t: real)

exception: none

output: ((ft = quafunction) ⇒fs_g_eqfunctionvalue(t)) |
 ((ft = cosfunction) ⇒fs_g_cosfunctionvalue(t)) |
 ((ft = expfunction) ⇒fs_g_expfunctionvalue(t))

D. 3.4 Displacement specification module

Prefix: ds_

Reference: MG – C. 3.1.7

D. 3.4.1 Interface syntax

Imported data types:

- **FunctionDefinitionT** from the experiment definition module
FunctionDefinitionT = set of FunctionTypeT
FunctionTypeT = {quafunction, expfunction, cosfunction}
- **TimeDefinitionT** from the experiment definition module
TimeDefinitionT = tuple of {inittime: real, timelength: real, timestep: real}

Exported data types:

```
DisplacementT = tuple of
  displu: Real*
  displv: Real*
  displw: Real*
  deltaxy: Real*          /* design for the future*/
  deltaxz: Real*          /* design for the future*/
  deltaxy: Real*          /* design for the future*/
  deltaxz: Real*          /* design for the future*/
}
```

External functions:

```
labelshow,          /* from the screen display module*/
listshow,           /* from the screen display module*/
buttonshow         /* from the screen display module */
fs_g_functionvalue /* from the function structure module */
ed_g_TimeDefinition /* from the experiment definition module */
ed_g_experimentclass /* from the experiment definition module*/
ed_g_experimenttype /* from the experiment definition module*/
```

Exported constants: none

Exported functions:

Routines name	Inputs	Outputs	Exception
ds s initdisp		DisplacementT	
ds s disputype	FunctionDefinitionT		
ds g displu	real		
ds s displvtype	FunctionDefinitionT		
ds g displv	real		
ds s displwtype	FunctionTypeT		
ds g displw	real		
ds s displu	Real*		
ds s displv	Real*		
ds s displw	Real*		

ds_g_displacement		DisplacementT	
ds_specifydisplacement			

D. 3.4.2 Interface semantics

Environmental variable:

labwindows: the graphic window where the experiment is displayed on the screen.

State variables:

utype, vtype, wtype: FunctionDefinitionT

disp: DisplacementT

State invariants: none

Assumptions:

- ds_s_initdisp() should be called before any other access routines.
- Function structure module and experiment definition module should be initialized before the displacement specification module.
- ds_g_dispu, ds_g_dispv, dispw are just designed for the displacement controlled experiment as well as ds_s_disputype, ds_s_dispvtype and ds_s_dispwttype.
- In the data type DisplacementT, deltaxy, deltaxz and deltaxz are designed for the future.

Access routines semantics:

ds_s_initdisp()

exception: None

transition/output: disp.dispu := *undefined*
disp.dispv := *undefined*
disp.dispw := *undefined*
out := disp

ds_s_disputype(u: FunctionDefinitionT)

exception: None

transition: utype := u

ds_s_dispvtype(v: FunctionDefinitionT)

exception: None

transition: vtype := v

ds_s_dispwttype(w: FunctionDefinitionT)

exception: none

transition: wtype := w

ds_g_dispu(t: real)

exception: (t > (time.init + time.length)) ⇒ (out_of_range)

transition: disp.dispu := fs_g_functionvalue(utype, t)

ds_g_dispv(t: real)

exception: (t > (time.init + time.length)) ⇒ (out_of_range)
 transition: disp.dispv := fs_g_functionvalue(vtype, t)

ds_g_dispw(t: real)
 exception: (t > (time.init + time.length)) ⇒ (out_of_range)
 transition: disp.dispw := fs_g_functionvalue(wtype, t)

ds_s_dispu(u: Real*)
 exception: none
 output: disp.dispu := u

ds_s_dispv(v: Real*)
 exception: none
 output: disp.dispv := v

ds_s_dispw(w: Real*)
 exception: none
 output: disp.dispw := w

ds_g_displacement()
 exception: none
 output: out := disp

ds_specifydisplacement()
 exception: none
 transition: Call the screen display module to set the environmental variable's state for the graphical interface of the displacement specification and then query the user to choose the displacement function type and set the state variables utype, vtype or wtype.

Event table:

Condition	Event	Action
When the 'Confirm' button is pressed.	@Click(Confirm)	class:= ed_g_experimentclass type:= ed_g_experimenttype Table D-2
When the 'Cancel' button is pressed	@Click(Cancel)	Do nothing

$(\text{class} = H_1) \wedge (\text{type} = H_2) \rightarrow G$

		uniaxial	biaxial	multiaxial	H ₂
H ₁	displacement-controlled	ds_s_initdisp ds_s_disputype	ds_s_initdisp ds_s_disputype ds_s_dispvtype	ds_s_initdisp ds_s_disputype ds_s_dispvtype ds_s_dispwtype	G
	load-controlled	ds_s_initdisp	ds_s_initdisp	ds_s_initdisp	

Table D-2 The order for the initialization

D. 3.5 Load specification module

Prefix: ls_

Reference: MG – C. 3.1.8

D. 3.5.1 Interface syntax

Imported data types:

- FunctionDefinitionT
FunctionDefinitionT = set of {quafunction, expfunction, cosfunction}
- TimeDefinitionT
TimeDefinitionT = tuple of {inittime: real, timelength: real, timestep: real}

Exported data types:

```
LoadT = tuple of
  loadu: Real*
  loadv: Real*
  loadw: Real*
  loaddeltaxy: Real* /* design for the future*/

  loaddeltayz: Real* /*design for the future */
  loaddeltaxz: Real* /*design for the future */
}
```

External functions:

```
labelshow /* from the screen display module*/
listshow /* from the screen display module*/
buttonshow /* from the screen display module*/
fs_g_functionvalue /* from the function structure module */
ed_g_experimentclass/* from the experiment definition module*/
ed_g_experimenttype /* from the experiment definition module*/
```

Exported constants: none

Exported functions:

Routines name	Inputs	Outputs	Exception
ls_s_initload		LoadT	
ls_s_loadutype	FunctionDefinitionT		
ls_g_loadu	real		
ls_s_loadvtype	FunctionDefinitionT		
ls_g_loadv	real		
ls_s_loadwtype	FunctionTypeT		
ls_g_loadw	real		
ls_g_load		LoadT	
ls_specifyload			

D. 3.5.2 Interface semantics

State variables:

utype, vtype, wtype: FunctionDefinitionT
load: LoadT

State invariants: none

Assumptions:

- Function structure module and experiment definition module should be initialized before the load specification module.
- In the data type LoadT, loaddeltaxy, loaddeltayz and loaddeltaxz are designed for the future.

Access routines semantics:

ls_s_initload()

exception: none
transition/output: load.loadu :=*undefined*, load.loadv := *undefined*, load.loadw :=*undefined*
out:= load

ls_s_loadutype(u: FunctionDefinitionT)

exception: none
transition: utype := u

ls_s_loadvtype(v: FunctionDefinitionT)

exception: none
transition: vtype := v

ls_s_loadwtype(w: FunctionDefinitionT)

exception: none
transition: wtype := w

ls_g_loadu(t: real)

exception: (t > (time.init + time.length)) ⇒ exc(out_of_range)
transition: load.loadu :=fs_g_functionvalue(utype, t)

ls_g_loadv(t: real)

exception: (t > (time.init + time.length)) ⇒ exc(out_of_range)
transition: load.loadv :=fs_g_functionvalue(vtype, t)

ls_g_loadw(t: real)

exception: (t > (time.init + time.length)) ⇒ exc(out_of_range)
transition: load.loadw :=fs_g_functionvalue(wtype, t)

ls_g_load()

exception: none

output: out := load

ls_specifyload()

exception: none

transition: Call the screen display module to set the environmental variable's state for the user graphical interface of the load specification and then query the user to choose the displacement function type and set the state variables utype, vtype or wtype.

Event table:

Condition	Event	Action
When the 'Confirm' button is pressed.	@Click(Confirm)	class:=ed_g_experimentclass type:=ed_g_experimenttype Table D-3
When the 'Cancel' button is pressed.	@Click(Cancel)	Do nothing

$(\text{class} = H_1) \wedge (\text{type} = H_2) \rightarrow G$

		uniaxial	biaxial	multiaxial	H ₂
H ₁	displacement-controlled	Do nothing	Do nothing	Do nothing	G
	load-controlled	ds_s_initdisp ls_s_initload ls_s_loadutype	ds_s_initdisp ls_s_initload ls_s_loadutype ls_s_loadvtype	ds_s_initdisp ls_s_initload ls_s_loadutype ls_s_loadvtype ls_s_loadwtype	

Table D-3 The order for the initialization

D. 3.6 Table structure module

Prefix: ts_

Reference: MG – C. 3.2.6

D. 3.6.1 Interface syntax

Imported data types:

TensorDataT from the tensor data definition module

TensorDataT = Sequence [DIM][DIM] of Real*

Exported data types:

ResultDataT = sequence of TensorDataT

Exported functions:

Routines name	Inputs	Outputs	Exception
ts_AddColumn	String	Integer	
ts_g_ColumnName	Integer	String	InvalidColumnNumber
ts_AddRow		Integer	
ts_RemoveRow	Integer		InvalidRowNumber
Ts_Depth		Integer	
Ts_Width		Integer	
ts_SetAt	Integer Integer ResultDataT		
ts_GetAt	Integer Integer	ResultDataT	
ts_RealseTable			

External Function:

String RealToString(resultdata:Real*)

Real* StringToReal(resultstring:string)

D. 3.6.2 Interface semantics

State variables:

resultData:ResultDataT

numCols,numRows: Integer

State invariants: none

Local variables:

ColumnNameList: sequence of String

resultstring: string

Local Functions:

ResultDataT CreatTable()

Assumption: none

Access routines semantics:

ts_AddColumn(columnname:String)

exception: None

transition/output: ColumnNameList[numCols] := columnname
 numCols := numCols+1
 out:=numCols

ts_g_ColumnName(index:integer)
 exception: (index > |ColumnNameList|)⇒InvalidColumnNumber
 transition/output: out:= ColumnNameList[index-1]

ts_AddRow()
 exception: none
 transition/output: resultData = CreatTable()
 Allocate memories for the resultdata
 numRows = numRows+1
 out:=numRows

ts_RemoveRow(rownum:integer)
 exception: (rownum > ts_Depth())⇒InvalidRowNumber
 transition: Release memories for this row
 numRows = numRows-1

ts_Depth()
 exception: none
 output: Out := numRows

ts_Width()
 exception: none
 transition: Out := numCols

ts_SetAt(row:integer,column:integer,result:ResultDataT)
 exception: (row > ts_Depth())⇒InvalidRowNumber
 (column > ts_Width())⇒ InvalidColumnNumber
 output: resultstring :=RealToString(result)
 allocate memory for resultstring
 resultData[row,column] := ResultString

ts_GetAt(row:integer,column:integer)
 exception: (row > ts_Depth())⇒InvalidRowNumber
 (column > ts_Width())⇒ InvalidColumnNumber
 transition: resultstring :=resultData[row,column]
 out:= StringToReal(resultstring)

Ts_ReleaseTable()
 exception: none

Output: CurrentRow :=ts_Depth
 CurrentColumn := ts_width
 Release the table in which the number of rows is currentRow
 and the number of colomns is currentColumn

D. 3.7 Screen display module

The screen display module is more related with the development environment. Exported functions might be provided by the development environment or as the operating system's application program interface. So there are not interface syntax and interface semantics for this module. The functions of access routines are just described in the following.

Environmental variable:

labwindows: the window where the experiment is displayed.

Exported functions

- `labelshow(top, left: integer, caption: string)`
At this position (top, left) a label caption is show.
- `optionshow(top, left: integer; visible: boolean; value: boolean; caption: string)`
At the position(top, left) an option is shown with the caption. The variable visible decides if it can be seen and variable value decides if this option is elected.
- `frameshow(top, left, down, right: integer; visible: boolean; caption: string)`
A frame with the caption is shown between the left top position (top, left) and right down position (down, right).
- `buttonshow(top, left, down, right: integer, visible: boolean, caption: string)`
A button with the caption is shown. The left top point of the button is at the coordinate (top, left) and right down point of this button is at the coordinate (down, right)
- `imshow(top, left, down, right: integer; imagefile: string)`
An imag is shown between the top left point (top, left) and bottom right point (down, right).
- `textshow(top, left, down, right: integer; visible: boolean; text: string)`
A text input window is shown between the top left point (top, left) and bottom right point (down, right).
- `checkboxshow(top, left: integer; visible: Boolean; value: Boolean; caption:string)`
At the position(top, left) a checkbox is shown with the caption. The variable visible decides if it can be seen and variable value decides if this option is elected.
- `listshow(top, left, down, right: integer; visible: Boolean)`
A listbox is shown between top left point (top, left) and bottom right point (down, right). The variable visible decides if it can be seen and variable value decides if this option is elected.
- `additemtolist(s: string)`
Add a new item to the list.
- `deleteitemtolist(i: integer)`
Delete the item with the specified index from the list.
- `showwindow`
Display the experiment window where the experiment is done.

D. 3.8 Experiment specification module

Prefix: es_

Reference: MG – C. 3.1.5

D. 3.8.1 Interface syntax

Environmental variables:

labwindows: the graphic window where the experiment is displayed

Exported data type: none

Exported constant: none

Exported function:

Routines name	Inputs	Outputs	Exception
es_specifyExperiment			

External functions:

labelshow /*from the screen display module*/

frameshow /* from the screen display module */

optionshow /* from the screen display module*/

buttonshow /*from screen display module*/

ed_s_expClass from the experiment definition module

ed_s_expType from the experiment definition module

D. 3.8.2 Interface semantics

State variables: none

State invariant: none

Assumption: none

Access routine semantics:

es_specifyexperiment

exception: none

Output: Call screen display module to configure the graphical interface and set the labwindows's state for the display and then query the user for the experiment class and type definition and store them in expClass and expType states respectively.

Event table:

Condition	Event	Action
When the 'Confirm' button is pressed.	@Click(Confirm)	ed_s_ExperimentClass(class) ed_s_ExperimentType(type)
When the 'Cancel' button	@Click(Cancel)	Do nothing

is pressed

D. 3.9 Function specification module

Prefix: fsm_

Reference: MG – C. 3.1.9

D. 3.9.1 Interface syntax

Environmental variables:

labwindows: the graphic window where the experiment is displayed.

Exported data type: none

Exported constant: none

Exported function:

Routines name	Inputs	Outputs	Exception
fsm_specifyfunction			

External function:

labelshow /* from the screen display module*/

textshow /* from the screen display module*/

checkboxshow/* from the screen display module*/

buttonshow /* from screen display module*/

real StringToReal(s: string)

return the real number by converting the string to the real number

fs_s_quafunction /*from the function structure module*/

fs_s_cosfunction /*from the function structure module*/

fs_s_expfunction /*from the function structure module*/

ed_g_functiondefinition /*from the experiment definition module*/

D. 3.9.2 Interface semantics

State variable: none

State invariant: none

Assumption: none

Access routine semantics:

fsm_specifyfunction

exception:

output:

Call screen display module to configure the graphical interface and set the environmental variable for the display and then query the user to select which functions will be used and set the selected functions' arguments and store them in states qua_state, con_state or exp_state respectively and set the corresponding state variables que_set, cos_set or exp_set as true.

Event table:

Condition	Event	Action
When the 'Confirm' button is pressed.	@Click(Confirm)	select := ed_g_functiondefinition Table D-4
When the 'Cancel' button is pressed	@Click(Cancel)	Do nothing

(select=H)→G

{}	Do nothing
{qua}	fs_s_quafunction is called.
{exp}	fs_s_expfunction is called.
{cos}	fs_s_cosfunction is called.
{qua, exp}	fs_s_quafunction and fs_s_expfunction are called.
{qua, cos}	fs_s_quafunction and fs_s_cosfunction are called.
{exp,cos}	fs_s_expfunction and fs_s_cosfunction are called.
{exp, cos, qua}	fs_s_expfunction, fs_s_cosfunction and fs_s_quafunction are called.

Table D-4 The order of initialization

Comments for the Table D-4:

- qua: the abbreviation of the quafunction
- exp: the abbreviation of the expfunction
- cos: the abbreviation of the cosfunction

D. 3.10 Specimen geometry specification module

Prefix: sgs_

Reference: MG – C. 3.1.6

D. 3.10.1 Interface syntax

Environmental variables:

labwindows: the graphic window, where the experiment is displayed.

Exported data type: none

Exported constant: none

Exported function:

Routines name	Inputs	Outputs	Exception
sgs_specifygeometry			

External functions:

labelshow /*from the screen display module*/

buttonshow /* from the screen display module*/

textshow /* from the screen display module */

real StringToReal(s: string)

Return the real number by converting the specified string in the textshow to the real type.

sg_s_geometrylength /* specimen geometry module*/

sg_s_geometrywidth /* specimen geometry module*/

sg_s_geometryheight /* specimen geometry module*/

D. 3.10.2 Interface semantics

State variable: none

State invariant: none

Assumption: none

Access routine semantics:

sgs_specifygeometry

exception: none

output: Call screen display module to configure the graphical user interface and set the environmental variable's state for the display and then query the user for the specimen geometry properties and store them in the state spe.

Event table:

Condition	Event	Action
When the 'Confirm' button is pressed.	@Click(Confirm)	ed_s_geometrylength ed_s_geometrywidth ed_s_geometryheight
When the 'Cancel' button is pressed.	@Click(Cancel)	Do nothing

Comments:

1. The length, width and height of the test specimen have to be specified in every experiment.

D. 3.11 Material properties file module

Prefix: mpf_

Reference: MG – C. 3.1.11

D. 3.11.1 Interface syntax

Exported data type:

```
PropertiesT = tuple of {  
    propertyname: string  
    propertyvalue: real  
}
```

PropertylistT: sequence of PropertiesT

PropertyValueT = sequence of real

Exported functions

Routines name	Inputs	Outputs	Exception
mpf_g_numberofproperties	string	integer	file_not_exist, failure_to_open
mpf_g_propertyname	string	PropertylistT	file_not_exist, failure_to_open
mpf_sg_propertyvalue	PropertyValueT	PropertylistT	

External functions:

Openfile(filename: string)

Open a file whose name is filename and a pointer pointing to this file.

Readfile(FILE *file, type: string)

Read the value of the specified type from an opened file. If type is string then read a string from the current position of the file pointer. If type is integer then read an integer value from the current position of the file pointer.

D. 3.11.2 Interface semantics

State variable:

f: file

State invariant: none

Assumption:

- Material property file is named with string given by a designer and an extension name (.mat). That string given by the designer should be consistent with constitutive equation file's name because every constitutive equation has the corresponding material properties saved as material property file. The difference between constitutive equation file and material property file is an extension. Material features file has an extension as .mat and constitutive equation file has an extension as .con.
- Material features file is written by a user with a notepad or textwriter and saved as a text. The order of file format is irrelevant, but it should be consistent so that the

order of information that is read from the file is the same as the order that is written to the file.

Local variables:

number: integer
list: PropertyListT

Access routine semantics:

mpf_g_numberofproperties(filename: string)
exception: (Openfile(filename) is unsuccessful) \Rightarrow (file_not_exist,
failure_to_open)
transition/output: f:= Openfile(filename)
out :=Readfile(f, "integer")

mpf_g_propertyname(filename: string)
exception: (Openfile(filename) is unsuccessful) \Rightarrow (file_not_exist,
failure_to_open)
transition/output: f:= Openfile(filename)
number :=Readfile(f, "integer")
for i=0 to number
list[i].propertyname := Readfile(f, "string")
out := list

mpf_sg_propertyvalue(value: PropertyValueT)
exception:
transition/output: f:= Openfile(filename)
number :=Readfile(f, "integer")
for i=0 to number
list[i].propertyvalue := value[i]
out := list

Comments:

1. The first entry in the material property file is the number of the material properties.

D. 3.12 Constitutive equation structure module

Prefix: cs_

Reference: MG – C. 3.2.1

D. 3.12.1 Interface syntax

Imported data type:

PropertyListT from the material property file module

```
PropertiesT = tuple of {  
    propertyname: string  
    propertyvalue: real  
}
```

ProptertylistT: sequence of PropertiesT

PropertyValueT from the material property file module

PropertyValueT = sequence of real

Exported data type:

DeformationListT = sequence[LISTNUM] of Boolean

```
ConstitutiveEquationT = tuple of {  
    name: string  
    the_number_of_material_properties: integer  
    material_properties_list: PropertyListT  
    deformation_list: DeformationListT  
}
```

Exported constant: none

Exported functions:

Routines name	Inputs	Outputs	Exception
cs_g_constitutiveequation		ConstitutiveEquationT	
cs_s_constitutiveequation	string DeformationListT PropertyValueT		
ce_g_writetofile	ConstitutiveEquationT		failure_to_open
ce_s_readfromfile	string	ConstitutiveEquationT	failure_to_open file_not_exist

External functions:

mpf_g_numberofproperties from the material property file module

mpf_g_propertyname from the material property file module

file Openfile(filename: string)

Open a file whose name is filename, if return value is zero, Opening a file is successful, otherwise failure.

Readfile(f: file, type: string)

Read the value of the specified type from an current position of a file pointer and return this value.

Writefile(f: file, var: type)

Write the value of a variable whose type is type constructor into the file.

D. 3.12.2 Interface semantics

State variable:

coninformation: ConstitutiveEquationT

f: file

Local variable:

local: PropertyListT

State invariant: none

Assumption:

- DeformationListT type is a sequence of boolean. This type is used to present if deformation definitions are used in the constitutive equation.
- Functions cs_g_readfile and cs_g_writetofile is used to operate the file in which the constitutive equation is saved. Actually the order of the format is irrelevant but it should be consistent so that the order of information on the constitutive equation that is read from the file is the same as the order that is written to the file.

Access routine semantics:

cs_s_constitutiveequation(name: string, deform: DeformationListT, value: PropertyValueT)

exception: none

transition: coninformation.name := filename

coninformation.the_number_of_material_properties:=
mpf_g_numberofproperties(name)

local := mpf_g_propertyname(name)

local := mpf_sg_propertyvalue(value)

coninformation.material_property_list := local

mpf_g_propertyname(name)

coninformation.deformation_list := deform

cs_g_constitutiveequation()

Exception: none

Output: out:= coninformation

cs_g_realdfromfile(filename: string)

Exception: (Openfile(filename) is unsuccessful) ⇒
(fail_to_open , file_not_exist)

Transition/output: f:= Openfile(filename)
local.name := filename
local.the_number_of_material_properties:=Readfile(f,"integer")
for i=0 to local.the_number_of_material_properties
 local.material_property_list[i] := Readfile(f, "PropertyT")
local.deformation_list := Readfile(f, "DeformationlistT")
out := local

cs_g_writetofile(conequ: ConstitutiveEquationT)

Exception: (Openfile(conequ.name) is unsuccessful) ⇒ fail_to_open

Transition/Output: f:= Openfile(conequ.name)

Writefile(f, conequ.the_number_of_material_properties)

For i=0 to conequ.the_number_of_material_properties

 Writefile(f, conequ.material_property_list[i])

Writefile(f, conequ.deformation_list)

D. 3.13 Constitutive equation specification module

Prefix: ces_

Reference: MG – C. 3.1.10

D. 3.13.1 Interface syntax

Environmental variable:

labwindows: the graphic window where the experiment is displayed.

Exported data type: none

Exported constant: none

Exported function:

Routines name	Inputs	Outputs	Exception
ces_specifyconstitutive			

External functions:

labelshow /* from the screen display module*/
textshow /* from the screen display module*/
frameshow /* from the screen display module*/
optionshow /* from the screen display module*/
buttonshow /* from the screen display module*/
listshow /* from the screen display module*/
ed_s_constitutiveswitch
ed_g_constitutiveswitch
cs_s_constitutiveequation
cs_g_readfromfile
cs_g_writetofile
mpf_g_numberofproperties
mpf_g_propertyname
mpf_g_sg_propertyvalue
real StringToReal

D. 3.13.2 Interface semantics

State variables: none

State invariant: none

Assumption: none

Access routine semantics:

ces_specifyconstitutive

Exception: None

Output: Call screen display module to set environmental variable 'labwindows' for the display and then query the user for the constitutive equation selection and keep it in con_switch state and query the user for the constitutive equation definition and keep it in the coninformation state and query the user for the value of material propterties.

Event table:

Condition	Event	Action
When the 'Confirm' button is pressed.	@Click(Confirm)	ed_s_constitutiveswitch, ce_s_constitutiveequation, mpf_g_numberofproperties, mpf_g_propertyname, mpf_sg_propertyvalue
When the 'Save' button is pressed	@Click(Save)	ce_g_writetofile
When the Cancel button is pressed	@Click(Cancel)	Do nothing

D. 3.14 Tensor data definition module

Prefix:

Reference:

D. 3.14.1 Interface syntax

Exported constant:

DIM	3
LISTNUM	15
MDG	1
SDG	2
MDPG	3
SDPG	4
CDT	5
GDT	6
ST	7
SRT	8
EFST	9
LFST	10
EIST	11
LIST	12
TST	13
TSST	14
ESST	15

Exported data type:

TensorDataT = sequence [DIM][DIM] of Real*

TensorFlagT = {mdg, sdg, mdpg, sdp, cdt, gdt, st, srt, efst, lfst, eist, list, tst, tsst, esst}

Comments:

- Comment on TensorFlagT
mdg: the abbreviation of material deformation gradient.
sdg: the abbreviation of spatial deformation gradient.
mdpg: the abbreviation of material displacement gradient.
sdp: the abbreviation of spatial displacement gradient.
cdt: the abbreviation of Cauchy's deformation gradient.
gdt: the abbreviation of Green's deformation gradient.
st: the abbreviation of stretch tensor.
srt: the abbreviation of stretch ratio tensor.
efst: the abbreviation of Eulerian finite strain tensor
lfst: the abbreviation of Lagrangian finite strain tensor
eist: the abbreviation of Eulerian infinitesimal strain tensor
list: the abbreviation of Lagrangian infinitesimal strain tensor
tst: the abbreviation of true strain tensor
tsst: the abbreviation of true stress tensor

esst : the abbreviation of engineering stress tensor.

D. 3.15 Material deformation gradient module

Prefix: mdg_

Reference: MG – C. 3.2.9

D. 3.15.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

DisplacementT = tuple of {dispu: real, dispv: real, dispw: real,
deltaxy: real, deltayz: real, deltaxz: real}

TensorDataT from Tensor data definition module

TensorDataT = sequence[DIM][DIM] of Real*

TensorFlagT = {mdg, sdg, mdpg, sdpg, cdt, gdt, st, srt, efst, lfst, eist, list, tst, tsst,
esst}

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
mdg_g_knownquantity	TensorDataT TensorFlagT	TensorDataT	Table D-5
mdg_g_geometry	SpecimenGeometryT DisplacementT	TensorDataT	

D. 3.15.2 Interface semantics

State variable: none

State invariant: none

Assumption:

Access routine semantics

mdg_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)

exception: Table D-7

output: Table D-5

mdg_g_geometry(sg: SpecimenGeometryT, disp:DisplacementT)
exception: None
output: Table D-6

H₁

	mdg	sdg	mdpg	sdpg	cdt	gdt	lfst	efst	list	eist	tst
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]=ud	kq	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]≠ud	kq	out[3][3] := 1/kq[3][3] others ud	out[3][3] := 1+kq[3][3] others ud.	out[3][3] := 1/(1-kq[3][3]) others ud.	out[3][3] := 1/sqrt(kq[3][3]) others ud.	out[3][3] := sqrt(kq[3][3]) others ud.	out[3][3] := sqrt(2*kq[3][3]+1) others ud.	out[3][3] := 1/sqrt(1-2*kq[3][3]) others ud.	out[3][3] := 1+kq[3][3] others ud.	out[3][3] := 1-1-kq[3][3] others ud.	out[3][3] := exp(kq[3][3]) others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]=ud	kq	out[2][2] := 1/kq[2][2] others ud	out[2][2] := 1+kq[2][2] others ud.	out[2][2] := 1/(1-kq[2][2]) others ud.	out[2][2] := 1/sqrt(kq[2][2]) others ud.	out[2][2] := sqrt(kq[2][2]) others ud.	out[2][2] := sqrt(2*kq[2][2]+1) others ud.	out[2][2] := 1/sqrt(1-2*kq[2][2]) others ud.	out[2][2] := 1+kq[2][2] others ud.	out[2][2] := 1-1-kq[2][2] others ud.	out[2][2] := exp(kq[2][2]) others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]≠ud	kq	out[2][2] := 1/kq[2][2] out[3][3] := 1/kq[3][3] others ud	out[2][2] := 1+kq[2][2] out[3][3] := 1+kq[3][3] others ud	out[2][2] := 1/(1-kq[2][2]) out[3][3] := 1/(1-kq[3][3]) others ud.	out[2][2] := 1/sqrt(kq[2][2]) out[3][3] := 1/sqrt(kq[3][3]) others ud.	out[2][2] := sqrt(kq[2][2]) out[3][3] := sqrt(kq[3][3]) others ud.	out[2][2] := sqrt(2*kq[2][2]+1) out[3][3] := sqrt(2*kq[3][3]+1) others ud.	out[2][2] := 1/sqrt(1-2*kq[2][2]) out[3][3] := 1/sqrt(1-2*kq[3][3]) others ud.	out[2][2] := 1+kq[2][2] out[3][3] := 1+kq[3][3] others ud	out[2][2] := 1-1-kq[2][2] out[3][3] := 1-1-kq[3][3] others ud.	out[2][2] := exp(kq[2][2]) out[3][3] := exp(kq[3][3]) others ud.
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]=ud	kq	out[1][1] := 1/kq[1][1] others ud	out[1][1] := 1+kq[1][1] others ud.	out[1][1] := 1/(1-kq[1][1]) others ud.	out[1][1] := 1/sqrt(kq[1][1]) others ud.	out[1][1] := sqrt(kq[1][1]) others ud.	out[1][1] := sqrt(2*kq[1][1]+1) others ud.	out[1][1] := 1/sqrt(1-2*kq[1][1]) others ud.	out[1][1] := 1+kq[1][1] others ud.	out[1][1] := 1/(1-kq[1][1]) others ud.	out[1][1] := exp(kq[1][1]) others ud.
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]≠ud	kq	out[1][1] := 1/kq[1][1] out[3][3] := 1/kq[3][3] others ud	out[1][1] := 1+kq[1][1] out[3][3] := 1+kq[3][3] others ud.	out[1][1] := 1/(1-kq[1][1]) out[3][3] := 1/(1-kq[3][3]) others ud.	out[1][1] := 1/sqrt(kq[1][1]) out[3][3] := 1/sqrt(kq[3][3]) others ud.	out[1][1] := sqrt(kq[1][1]) out[3][3] := sqrt(kq[3][3]) others ud.	out[1][1] := sqrt(2*kq[1][1]+1) out[3][3] := sqrt(2*kq[3][3]+1) others ud.	out[1][1] := 1/sqrt(1-2*kq[1][1]) out[3][3] := 1/sqrt(1-2*kq[3][3]) others ud.	out[1][1] := 1+kq[1][1] out[3][3] := 1+kq[3][3] others ud.	out[1][1] := 1/(1-kq[1][1]) out[3][3] := 1/(1-kq[3][3]) others ud.	out[1][1] := exp(kq[1][1]) out[3][3] := exp(kq[3][3]) others ud.
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]=ud	kq	out[1][1] := 1/kq[1][1] out[2][2] := 1/kq[2][2] others ud	out[1][1] := 1+kq[1][1] out[2][2] := 1+kq[2][2] others ud.	out[1][1] := 1/(1-kq[1][1]) out[2][2] := 1/(1-kq[2][2]) others ud.	out[1][1] := 1/sqrt(kq[1][1]) out[2][2] := 1/sqrt(kq[2][2]) others ud.	out[1][1] := sqrt(kq[1][1]) out[2][2] := sqrt(kq[2][2]) others ud.	out[1][1] := sqrt(2*kq[1][1]+1) out[2][2] := sqrt(2*kq[2][2]+1) others ud.	out[1][1] := 1/sqrt(1-2*kq[1][1]) out[2][2] := 1/sqrt(1-2*kq[2][2]) others ud.	out[1][1] := 1+kq[1][1] out[2][2] := 1+kq[2][2] others ud.	out[1][1] := 1/(1-kq[1][1]) out[2][2] := 1/(1-kq[2][2]) others ud.	out[1][1] := exp(kq[1][1]) out[2][2] := exp(kq[2][2]) others ud.
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]≠ud	kq	out[1][1] := 1/kq[1][1] out[2][2] := 1/kq[2][2] out[3][3] := 1/kq[3][3] others ud	out[1][1] := 1+kq[1][1] out[2][2] := 1+kq[2][2] out[3][3] := 1+kq[3][3] others ud.	out[1][1] := 1/(1-kq[1][1]) out[2][2] := 1/(1-kq[2][2]) out[3][3] := 1/(1-kq[3][3]) others ud.	out[1][1] := 1/sqrt(kq[1][1]) out[2][2] := 1/sqrt(kq[2][2]) out[3][3] := 1/sqrt(kq[3][3]) others ud.	out[1][1] := sqrt(kq[1][1]) out[2][2] := sqrt(kq[2][2]) out[3][3] := sqrt(kq[3][3]) others ud.	out[1][1] := sqrt(2*kq[1][1]+1) out[2][2] := sqrt(2*kq[2][2]+1) out[3][3] := sqrt(2*kq[3][3]+1) others ud.	out[1][1] := 1/sqrt(1-2*kq[1][1]) out[2][2] := 1/sqrt(1-2*kq[2][2]) out[3][3] := 1/sqrt(1-2*kq[3][3]) others ud.	out[1][1] := 1+kq[1][1] out[2][2] := 1+kq[2][2] out[3][3] := 1+kq[3][3] others ud.	out[1][1] := 1/(1-kq[1][1]) out[2][2] := 1/(1-kq[2][2]) out[3][3] := 1/(1-kq[3][3]) others ud.	out[1][1] := exp(kq[1][1]) out[2][2] := exp(kq[2][2]) out[3][3] := exp(kq[3][3]) others ud.

G

Table D-5 out for mdg_g_knownquantity

H₁ ∧ H₂ → G

H

disp.dispu=ud^ disp.dispv=ud^ disp.dispw=ud	Out :=[ud]
disp.dispu=ud^ disp.dispv=ud^ disp.dispw≠ud	out[3][3] := (1 + disp.dispw / sg.width) others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw=ud	out[2][2] := (1 + disp.dispv / sg.height) others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw≠ud	out[2][2] := (1 + disp.dispv / sg.height) out[3][3] := (1 + disp.dispw / sg.width) others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw=ud	out[1][1] := (1 + disp.dispu / sg.length) others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw≠ud	out[1][1] := (1 + disp.dispu / sg.length) out[3][3] := (1 + disp.dispw / sg.width) others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw=ud	out[1][1] := (1 + disp.dispu / sg.length) out[2][2] := (1 + disp.dispv / sg.height) others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw≠ud	out[1][1] := (1 + disp.dispu / sg.length) out[2][2] := (1 + disp.dispv / sg.height) out[3][3] := (1 + disp.dispw / sg.width) others ud

Table D-6 out for mdg_g_geometry

H_1

mdg	sdg	mdpg	sdpg	cdt	gdt	lfst	efst	list	eist	tst
non	((kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0)) ⇒d_zero	non	((kq[1][1]=1)∨ (kq[2][2]=1)∨ (kq[3][3]=1)) ⇒d_zero	((kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0)) ⇒d_zero, ((kq[1][1]<0)∨ (kq[2][2]<0)∨ (kq[3][3]<0)) ⇒sr_zero	((kq[1][1]<0)∨ (kq[2][2]<0)∨ (kq[3][3]<0)) ⇒sr_zero	((kq[1][1]<0.5)∨ (kq[2][2]<0.5)∨ (kq[3][3]<0.5)) ⇒sr_zero	((kq[1][1]>0.5)∨ (kq[2][2]>0.5)∨ (kq[3][3]>0.5)) ⇒sr_zero	non	((kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0)) ⇒d_zero	non

G

Table D-7 exception for mdg_g_knownquantity

 $H_1 \rightarrow G$

D. 3.16 Spatial deformation gradient module

Prefix: `sdg_`

Reference: MG C. 3.2.10

D. 3.16.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

TensorFlagT from the tensor data definition module

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
<code>sdg_g_knownquantity</code>	TensorDataT TensorFlagT	TensorDataT	TableD-8
<code>sdg_g_geometry</code>	SpecimenGeometryT DisplacementT	TensorDataT	none

D. 3.16.2 Interface semantics

State variable: none

State invariant: none

Assumption:

Access routine semantics

`sdg_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)`

exception: Table D-8

output: Table D-9

`sdg_g_geometry(sg: SpecimenGeometryT, disp: DisplacementT)`

Exception: None

Output: Table D-10

H_1	mdg	sdg	mdpg	sdpd	cdt	gdt	lfst	efst	list	eist	tst
G	$(kq[1][1]=0) \vee$ $(kq[2][2]=0) \vee$ $(kq[3][3]=0)$ \Rightarrow d_zero	non	$(kq[1][1]=1) \vee$ $(kq[2][2]=1) \vee$ $(kq[3][3]=1)$ \Rightarrow d_zero	non	$(kq[1][1]<0) \vee$ $(kq[2][2]<0) \vee$ $(kq[3][3]<0)$ \Rightarrow sr_lesszero	$(kq[1][1]<0) \vee$ $(kq[2][2]<0) \vee$ $(kq[3][3]<0)$ \Rightarrow sr_lesszero, $(kq[1][1]=0) \vee$ $(kq[2][2]=0) \vee$ $(kq[3][3]=0)$ \Rightarrow d_zero	$(kq[1][1]<-$ 0.5) \vee $(kq[2][2]<-$ 0.5) \vee $(kq[3][3]<-$ 0.5) \Rightarrow sr_lesszero, $(kq[1][1]=-$ 0.5) \vee $(kq[2][2]=-$ 0.5) \vee $(kq[3][3]=-$ 0.5) \Rightarrow d_zero	$(kq[1][1]>0.5) \vee$ $(kq[2][2]>0.5) \vee$ $(kq[3][3]>0.5)$ \Rightarrow sr_lesszero	$(kq[1][1]=-1)$ \vee $(kq[2][2]=-1)$ \vee $(kq[3][3]=-1)$ \Rightarrow d_zero	non	non

$H_1 \rightarrow G$

Table D-8 exception for sdg_g_knownquantity

H₂ : kgflag

H₁

	mdg	sdg	mdpg	sdpg	cdt	gdt
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	Out:=[ud]	kq	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := 1 / kq[3][3]$ others ud	kq	$out[3][3] := 1 / (1 + kq[3][3])$ others ud.	$out[3][3] := 1 - kq[3][3]$ others ud.		$out[3][3] := 1 / \sqrt{kq[3][3]}$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[2][2] := 1 / kq[2][2]$ others ud	kq	$out[2][2] := 1 / (1 + kq[2][2])$ others ud.	$out[2][2] := 1 - kq[2][2]$ others ud.	$out[2][2] := \sqrt{kq[2][2]}$ others ud.	$out[2][2] := 1 / \sqrt{kq[2][2]}$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := 1 / kq[2][2]$ $out[3][3] := 1 / kq[3][3]$ others ud	kq	$out[2][2] := 1 / (1 + kq[2][2])$ $out[3][3] := 1 / (1 + kq[3][3])$ others ud	$out[2][2] := 1 - kq[2][2]$ $out[3][3] := 1 - kq[3][3]$ others ud.	$out[2][2] := \sqrt{kq[2][2]}$ $out[3][3] := \sqrt{kq[3][3]}$ others ud.	$out[2][2] := 1 / \sqrt{kq[2][2]}$ $out[3][3] := 1 / \sqrt{kq[3][3]}$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := 1 / kq[1][1]$ others ud	kq	$out[1][1] := 1 / (1 + kq[1][1])$ others ud.	$out[1][1] := 1 - kq[1][1]$ others ud.	$out[1][1] := \sqrt{kq[1][1]}$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]}$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := 1 / kq[1][1]$ $out[3][3] := 1 / kq[3][3]$ others ud	kq	$out[1][1] := 1 / (1 + kq[1][1])$ $out[3][3] := 1 / (1 + kq[3][3])$ others ud.	$out[1][1] := 1 - kq[1][1]$ $out[3][3] := 1 - kq[3][3]$ others ud.	$out[1][1] := \sqrt{kq[1][1]}$ $out[3][3] := \sqrt{kq[3][3]}$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]}$ $out[3][3] := 1 / \sqrt{kq[3][3]}$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := 1 / kq[1][1]$ $out[2][2] := 1 / kq[2][2]$ others ud	kq	$out[1][1] := 1 / (1 + kq[1][1])$ $out[2][2] := 1 / (1 + kq[2][2])$ others ud.	$out[1][1] := 1 - kq[1][1]$ $out[2][2] := 1 - kq[2][2]$ others ud.	$out[1][1] := \sqrt{kq[1][1]}$ $out[2][2] := \sqrt{kq[2][2]}$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]}$ $out[2][2] := 1 / \sqrt{kq[2][2]}$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := 1 / kq[1][1]$ $out[2][2] := 1 / kq[2][2]$ $out[3][3] := 1 / kq[3][3]$ others ud	Kq	$out[1][1] := 1 / (1 + kq[1][1])$ $out[2][2] := 1 / (1 + kq[2][2])$ $out[3][3] := 1 / (1 + kq[3][3])$ others ud.	$out[1][1] := 1 - kq[1][1]$ $out[2][2] := 1 - kq[2][2]$ $out[3][3] := 1 - kq[3][3]$ others ud.	$out[1][1] := \sqrt{kq[1][1]}$ $out[2][2] := \sqrt{kq[2][2]}$ $out[3][3] := \sqrt{kq[3][3]}$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]}$ $out[2][2] := 1 / \sqrt{kq[2][2]}$ $out[3][3] := 1 / \sqrt{kq[3][3]}$ others ud.

G

Table D-9 out for sdg_g_knownquantity

H₁ ∧ H₂ → G

Continue→

$H_2 : \text{kgflag}$

H_1

	lfst	efst	list	eist	tst
	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$
$kq[1][1] = \text{ud} \wedge kq[2][2] = \text{ud} \wedge kq[3][3] = \text{ud}$	$\text{out}[3][3] := 1 / \text{sqrt}(2 * kq[3][3] + 1)$ others ud.	$\text{out}[3][3] := \text{sqrt}(1 - 2 * kq[3][3])$ others ud.	$\text{out}[3][3] := 1 / (1 + kq[3][3])$ others ud.	$\text{out}[3][3] := 1 - kq[3][3]$ others ud.	$\text{out}[3][3] := \exp(-kq[3][3])$ others ud.
$kq[1][1] = \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] = \text{ud}$	$\text{out}[2][2] := 1 / \text{sqrt}(2 * kq[2][2] + 1)$ others ud.	$\text{out}[2][2] := \text{qrt}(1 - 2 * kq[2][2])$ others ud.	$\text{out}[2][2] := 1 / (1 + kq[2][2])$ others ud.	$\text{out}[2][2] := 1 - kq[2][2]$ others ud.	$\text{out}[2][2] := \exp(-kq[2][2])$ others ud.
$kq[1][1] = \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[2][2] := 1 / \text{sqrt}(2 * kq[2][2] + 1)$ $\text{out}[3][3] := 1 / \text{sqrt}(2 * kq[3][3] + 1)$ others ud.	$\text{out}[2][2] := \text{sqrt}(1 - 2 * kq[2][2])$ $\text{out}[3][3] := \text{sqrt}(1 - 2 * kq[3][3])$ others ud.	$\text{out}[2][2] := 1 / (1 + kq[2][2])$ $\text{out}[3][3] := 1 / (1 + kq[3][3])$ others ud.	$\text{out}[2][2] := 1 - kq[2][2]$ $\text{out}[3][3] := 1 - kq[3][3]$ others ud.	$\text{out}[2][2] := \exp(-kq[2][2])$ $\text{out}[3][3] := \exp(-kq[3][3])$ others ud.
$kq[1][1] \neq \text{ud} \wedge kq[2][2] = \text{ud} \wedge kq[3][3] = \text{ud}$	$\text{out}[1][1] := 1 / \text{sqrt}(2 * kq[1][1] + 1)$ others ud.	$\text{out}[1][1] := \text{sqrt}(1 - 2 * kq[1][1])$ others ud.	$\text{out}[1][1] := 1 / (1 + kq[1][1])$ others ud.	$\text{out}[1][1] := 1 - kq[1][1]$ others ud.	$\text{out}[1][1] := \exp(-kq[1][1])$ others ud.
$kq[1][1] \neq \text{ud} \wedge kq[2][2] = \text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[1][1] := 1 / \text{sqrt}(2 * kq[1][1] + 1)$ $\text{out}[3][3] := 1 / \text{sqrt}(2 * kq[3][3] + 1)$ others ud.	$\text{out}[1][1] := \text{sqrt}(1 - 2 * kq[1][1])$ $\text{out}[3][3] := \text{sqrt}(1 - 2 * kq[3][3])$ others ud.	$\text{out}[1][1] := 1 / (1 + kq[1][1])$ $\text{out}[3][3] := 1 / (1 + kq[3][3])$ others ud.	$\text{out}[1][1] := 1 - kq[1][1]$ $\text{out}[3][3] := 1 - kq[3][3]$ others ud.	$\text{out}[1][1] := \exp(-kq[1][1])$ $\text{out}[3][3] := \exp(-kq[3][3])$ others ud.
$kq[1][1] \neq \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] = \text{ud}$	$\text{out}[1][1] := 1 / \text{sqrt}(2 * kq[1][1] + 1)$ $\text{out}[2][2] := 1 / \text{sqrt}(2 * kq[2][2] + 1)$ others ud.	$\text{out}[1][1] := \text{sqrt}(1 - 2 * kq[1][1])$ $\text{out}[2][2] := \text{sqrt}(1 - 2 * kq[2][2])$ others ud.	$\text{out}[1][1] := 1 / (1 + kq[1][1])$ $\text{out}[2][2] := 1 / (1 + kq[2][2])$ others ud.	$\text{out}[1][1] := 1 - kq[1][1]$ $\text{out}[2][2] := 1 - kq[2][2]$ others ud.	$\text{out}[1][1] := \exp(-kq[1][1])$ $\text{out}[2][2] := \exp(-kq[2][2])$ others ud.
$kq[1][1] \neq \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[1][1] := 1 / \text{sqrt}(2 * kq[1][1] + 1)$ $\text{out}[2][2] := 1 / \text{sqrt}(2 * kq[2][2] + 1)$ $\text{out}[3][3] := 1 / \text{sqrt}(2 * kq[3][3] + 1)$ others ud.	$\text{out}[1][1] := \text{sqrt}(1 - 2 * kq[1][1])$ $\text{out}[2][2] := \text{sqrt}(1 - 2 * kq[2][2])$ $\text{out}[3][3] := \text{sqrt}(1 - 2 * kq[3][3])$ others ud.	$\text{out}[1][1] := 1 / (1 + kq[1][1])$ $\text{out}[2][2] := 1 / (1 + kq[2][2])$ $\text{out}[3][3] := 1 / (1 + kq[3][3])$ others ud.	$\text{out}[1][1] := 1 - kq[1][1]$ $\text{out}[2][2] := 1 - kq[2][2]$ $\text{out}[3][3] := 1 - kq[3][3]$ others ud.	$\text{out}[1][1] := \exp(-kq[1][1])$ $\text{out}[2][2] := \exp(-kq[2][2])$ $\text{out}[3][3] := \exp(-kq[3][3])$ others ud.

→Continue

G

Table D-9 out for $\text{sdg_g_knownquantity}$ (Continue)

$H_1 \wedge H_2 \rightarrow G$

G

H→G

H	disp.dispu=ud∧ disp.dispv=ud∧ disp.dispw=ud	Out :=[ud]
	disp.dispu=ud∧ disp.dispv=ud∧ disp.dispw≠ud	out[3][3] := (1 + disp.dispw / sg.width) ⁻¹ others ud
	disp.dispu=ud∧ disp.dispv≠ud∧ disp.dispw=ud	out[2][2] := (1 + disp.dispv / sg.height) ⁻¹ others ud
	disp.dispu=ud∧ disp.dispv≠ud∧ disp.dispw≠ud	out[2][2] := (1 + disp.dispv / sg.height) ⁻¹ out[3][3] := (1 + disp.dispw / sg.width) ⁻¹ others ud
	disp.dispu≠ud∧ disp.dispv=ud∧ disp.dispw=ud	out[1][1] := (1 + disp.dispu / sg.length) ⁻¹ others ud
	disp.dispu≠ud∧ disp.dispv=ud∧ disp.dispw≠ud	out[1][1] := (1 + disp.dispu / sg.length) ⁻¹ out[3][3] := (1 + disp.dispw / sg.width) ⁻¹ others ud
	disp.dispu≠ud∧ disp.dispv≠ud∧ disp.dispw=ud	out[1][1] := (1 + disp.dispu / sg.length) ⁻¹ out[2][2] := (1 + disp.dispv / sg.height) ⁻¹ others ud
	disp.dispu≠ud∧ disp.dispv≠ud∧ disp.dispw≠ud	out[1][1] := (1 + disp.dispu / sg.length) ⁻¹ out[2][2] := (1 + disp.dispv / sg.height) ⁻¹ out[3][3] := (1 + disp.dispw / sg.width) ⁻¹ others ud

Table D-10 out for sdg_g_geometry

D. 3.17 Material displacement gradient module

Prefix: mdpg_

Reference: MG C. 3.2.11

D. 3.17.1 Interface syntax

Imported data type:

TensorDataT from the material deformation gradient

TensorFlagT from the material deformation gradient

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
mdpg_g_knownquantity	TensorDataT TensorFlagT	TensorDataT	
mdpg_g_geometry	SpecimenGeometryT DisplacementT	TensorDataT	none

D. 3.17.2 Interface semantics

State variable: none

State invariant: none

Assumption:

Access routine semantics

mdpg_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)

exception: Table D-11

output: Table D-12

mdpg_g_geometry(sg: SpecimenGeometryT, disp: DisplacementT)

Exception: None

Output: Table D-13

H_1	mdg	sdg	mdpg	sdpq	cdt	gdt	lfst	efst	list	eist	tst
G	non	(kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0) ⇒ d_zero	non	(kq[1][1]=1)∨ (kq[2][2]=1)∨ (kq[3][3]=1) ⇒ d_zero	(kq[1][1]<0)∨ (kq[2][2]<0)∨ (kq[3][3]<0) ⇒ sr_lesszero (kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0) ⇒ d_zero	(kq[1][1]<0)∨ (kq[2][2]<0)∨ (kq[3][3]<0) ⇒ sr_lesszero,	(kq[1][1]<- 0.5)∨ (kq[2][2]<- 0.5)∨ (kq[3][3]<- 0.5) ⇒ sr_lesszero,	(kq[1][1]>0.5)∨ (kq[2][2]>0.5)∨ (kq[3][3]>0.5) ⇒ sr_lesszero, (kq[1][1]=0.5)∨ (kq[2][2]=0.5)∨ (kq[3][3]=0.5) ⇒ d_zero	non	(kq[1][1]=1) ∨ (kq[2][2]=1) ∨ (kq[3][3]=1) ⇒ d_zero	non

$H_1 \rightarrow G$

Table D-11 exception for mdpq_g_knownquantity

$H_2 : \text{kgflag}$

H_1

	mdg	sdg	mdpg	sdpg	cdt	gdt
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	Out:=[ud]	Out:=[ud]	kq	Out:=[ud]	Out:=[ud]	Out:=[ud]
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := kq[3][3] - 1$ others ud	$out[3][3] := 1 / kq[3][3] - 1$ others ud	kq	$out[3][3] := kq[3][3] / (1 - kq[3][3])$ others ud.	$out[3][3] := 1 / \sqrt{kq[3][3]} - 1$ others ud	$out[3][3] := \sqrt{kq[3][3]} - 1$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$kq[2][2] := kq[2][2] - 1$ others ud	$out[2][2] := 1 / kq[2][2] - 1$ others ud	kq	$out[2][2] := kq[2][2] / (1 - kq[2][2])$ others ud.	$out[2][2] := 1 / \sqrt{kq[2][2]} - 1$ others ud	$out[2][2] := \sqrt{kq[2][2]} - 1$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := kq[2][2] - 1$ $out[3][3] := kq[3][3] - 1$ others ud	$out[2][2] := 1 / kq[2][2] - 1$ $out[3][3] := 1 / kq[3][3] - 1$ others ud	kq	$out[2][2] := kq[2][2] / (1 - kq[2][2])$ $out[3][3] := kq[3][3] / (1 - kq[3][3])$ others ud.	$out[2][2] := 1 / \sqrt{kq[2][2]} - 1$ $out[3][3] := 1 / \sqrt{kq[3][3]} - 1$ others ud	$out[2][2] := \sqrt{kq[2][2]} - 1$ $out[3][3] := \sqrt{kq[3][3]} - 1$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := kq[1][1] - 1$ others ud	$out[1][1] := 1 / kq[1][1] - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]} - 1$ others ud	$out[1][1] := \sqrt{kq[1][1]} - 1$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := kq[1][1] - 1$ $out[3][3] := kq[3][3] - 1$ others ud	$out[1][1] := 1 / kq[1][1] - 1$ $out[3][3] := 1 / kq[3][3] - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ $out[3][3] := kq[3][3] / (1 - kq[3][3])$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]} - 1$ $out[3][3] := 1 / \sqrt{kq[3][3]} - 1$ others ud	$out[1][1] := \sqrt{kq[1][1]} - 1$ $out[3][3] := \sqrt{kq[3][3]} - 1$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := kq[1][1] - 1$ $out[2][2] := kq[2][2] - 1$ others ud	$out[1][1] := 1 / kq[1][1] - 1$ $out[2][2] := 1 / kq[2][2] - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ $out[2][2] := kq[2][2] / (1 - kq[2][2])$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]} - 1$ $out[2][2] := 1 / \sqrt{kq[2][2]} - 1$ others ud	$out[1][1] := \sqrt{kq[1][1]} - 1$ $out[2][2] := \sqrt{kq[2][2]} - 1$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := kq[1][1] - 1$ $out[2][2] := kq[2][2] - 1$ $out[3][3] := kq[3][3] - 1$ others ud	$out[1][1] := 1 / kq[1][1] - 1$ $out[2][2] := 1 / kq[2][2] - 1$ $out[3][3] := 1 / kq[3][3] - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ $out[2][2] := kq[2][2] / (1 - kq[2][2])$ $out[3][3] := kq[3][3] / (1 - kq[3][3])$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]} - 1$ $out[2][2] := 1 / \sqrt{kq[2][2]} - 1$ $out[3][3] := 1 / \sqrt{kq[3][3]} - 1$ others ud	$out[1][1] := \sqrt{kq[1][1]} - 1$ $out[2][2] := \sqrt{kq[2][2]} - 1$ $out[3][3] := \sqrt{kq[3][3]} - 1$ others ud

G

Table D-12 out for mdpg_g_knownquantity

$H_1 \wedge H_2 \rightarrow G$

Continue →

$H_2 : \text{kgflag}$

H_1

	lfst	efst	list	eist	tst
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	Out:=[ud]	Out:=[ud]	kq	Out:=[ud]	Out:=[ud]
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := \text{sqrt}(2 * kq[3][3] + 1) - 1$ others ud.	$out[3][3] := 1 / \text{sqrt}(1 - 2 * kq[3][3]) - 1$ others ud	kq	$out[3][3] := kq[3][3] / (1 - kq[3])$ others ud.	$out[3][3] := \exp(kq[3][3]) - 1$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[2][2] := \text{sqrt}(2 * kq[2][2] + 1) - 1$ others ud.	$out[2][2] := 1 / \text{sqrt}(1 - 2 * kq[2][2]) - 1$ others ud	kq	$out[2][2] := kq[2][2] / (1 - kq[2])$ others ud.	$out[2][2] := \exp(kq[2][2]) - 1$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := \text{sqrt}(2 * kq[2][2] + 1) - 1$ $out[3][3] := \text{sqrt}(2 * kq[3][3] + 1) - 1$ others ud.	$out[2][2] := 1 / \text{sqrt}(1 - 2 * kq[2][2]) - 1$ $out[3][3] := 1 / \text{sqrt}(1 - 2 * kq[3][3]) - 1$ others ud	kq	$out[2][2] := kq[2][2] / (1 - kq[2])$ $out[3][3] := kq[3][3] / (1 - kq[3])$ others ud.	$out[2][2] := \exp(kq[2][2]) - 1$ $out[3][3] := \exp(kq[3][3]) - 1$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := \text{sqrt}(2 * kq[1][1] + 1) - 1$ others ud.	$out[1][1] := 1 / \text{sqrt}(1 - 2 * kq[1][1]) - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ others ud.	$out[1][1] := \exp(kq[1][1]) - 1$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := \text{sqrt}(2 * kq[1][1] + 1) - 1$ $out[3][3] := \text{sqrt}(2 * kq[3][3] + 1) - 1$ others ud.	$out[1][1] := 1 / \text{sqrt}(1 - 2 * kq[1][1]) - 1$ $out[3][3] := 1 / \text{sqrt}(1 - 2 * kq[3][3]) - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ $out[3][3] := kq[3][3] / (1 - kq[3])$ others ud.	$out[1][1] := \exp(kq[1][1]) - 1$ $out[3][3] := \exp(kq[3][3]) - 1$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := \text{sqrt}(2 * kq[1][1] + 1) - 1$ $out[2][2] := \text{sqrt}(2 * kq[2][2] + 1) - 1$ others ud.	$out[1][1] := 1 / \text{sqrt}(1 - 2 * kq[1][1]) - 1$ $out[2][2] := 1 / \text{sqrt}(1 - 2 * kq[2][2]) - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ $out[2][2] := kq[2][2] / (1 - kq[2])$ others ud.	$out[1][1] := \exp(kq[1][1]) - 1$ $out[2][2] := \exp(kq[2][2]) - 1$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := \text{sqrt}(2 * kq[1][1] + 1) - 1$ $out[2][2] := \text{sqrt}(2 * kq[2][2] + 1) - 1$ $out[3][3] := \text{sqrt}(2 * kq[3][3] + 1) - 1$ others ud.	$out[1][1] := 1 / \text{sqrt}(1 - 2 * kq[1][1]) - 1$ $out[2][2] := 1 / \text{sqrt}(1 - 2 * kq[2][2]) - 1$ $out[3][3] := 1 / \text{sqrt}(1 - 2 * kq[3][3]) - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ $out[2][2] := kq[2][2] / (1 - kq[2])$ $out[3][3] := kq[3][3] / (1 - kq[3])$ others ud.	$out[1][1] := \exp(kq[1][1]) - 1$ $out[2][2] := \exp(kq[2][2]) - 1$ $out[3][3] := \exp(kq[3][3]) - 1$ others ud.

→Continue

G

Table D-12 out for mdpg_g_knownquantity (Continue)

$H_1 \wedge H_2 \rightarrow G$

G

H→G

H

disp.dispu=ud^ disp.dispv=ud^ disp.dispw=ud	Out :=[ud]
disp.dispu=ud^ disp.dispv=ud^ disp.dispw≠ud	<i>out[3][3] := disp.dispw / sg.width</i> others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw=ud	<i>out[2][2] := disp.dispv / sg.height</i> others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw≠ud	<i>out[2][2] := disp.dispv / sg.height</i> <i>out[3][3] := disp.dispw / sg.width</i> others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw=ud	<i>out[1][1] := disp.dispu / sg.length</i> others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw≠ud	<i>out[1][1] := disp.dispu / sg.length</i> <i>out[3][3] := disp.dispw / sg.width</i> others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw=ud	<i>out[1][1] := disp.dispu / sg.length</i> <i>out[2][2] := disp.dispv / sg.height</i> others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw≠ud	<i>out[1][1] := disp.dispu / sg.length</i> <i>out[2][2] := disp.dispv / sg.height</i> <i>out[3][3] := disp.dispw / sg.width</i> others ud

Table D-13 out for mdp_g_g_geometry

D. 3.18 Spatial displacement gradient module

Prefix: `sdpg_`

Reference: MG C. 3.2.12

D. 3.18.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

TensorFlagT from the tensor data definition module

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
<code>sdpg_g_knownquantity</code>	TensorDataT TensorFlagT	TensorDataT	Table D-14
<code>sdpg_g_geometry</code>	SpecimenGeometryT DisplacementT	TensorDataT	none

D. 3.18.2 Interface semantics

State variable: none

State invariant: none

Assumption:

Access routine semantics

`sdpg_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)`

exception: Table D-14

output: Table D-15

`sdpg_g_geometry(sg: SpecimenGeometryT, disp: DisplacementT)`

exception: none

Output: Table D-16

H_1	mdg	sdg	mdpg	sdpg	cdt	gdt	lfst	efst	list	eist	tst
G	(kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0) ⇒ d_zero	non	(kq[1][1]=1)∨ (kq[2][2]=1)∨ (kq[3][3]=1) ⇒ d_zero	non	(kq[1][1]<0) ∨ (kq[2][2]<0) ∨ (kq[3][3]<0) ⇒ sr_lesszero	(kq[1][1]<0) ∨ (kq[2][2]<0) ∨ (kq[3][3]<0) ⇒ sr_lesszero, (kq[1][1]=0) ∨ (kq[2][2]=0) ∨ (kq[3][3]=0) ⇒ d_zero	(kq[1][1]<-0.5) ∨ (kq[2][2]<-0.5) ∨ (kq[3][3]<-0.5) ⇒ sr_lesszero, (kq[1][1]=-0.5) ∨ (kq[2][2]=-0.5) ∨ (kq[3][3]=-0.5) ⇒ d_zero	(kq[1][1]>0.5) ∨ (kq[2][2]>0.5) ∨ (kq[3][3]>0.5) ⇒ sr_lesszero,	(kq[1][1]=-1) ∨ (kq[2][2]=-1) ∨ (kq[3][3]=-1) ⇒ d_zero non	non	non

$H_1 \rightarrow G$

Table D-14 exception for sdpg_g_knownquantity

H_1

	mdg	sdg	mdpg	sdpg	cdt	gdt
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	Out:={ud}	Out:={ud}	Out:={ud}	kq	Out:={ud}	Out:={ud}
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := 1 - 1/kq[3][3]$ others ud	$out[3][3] := 1 - kq[3][3]$ others ud	$out[3][3] := kq[3][3]/(1 + kq[3][3])$ others ud.	kq	$out[3][3] := 1 - \text{sqrt}(kq[3][3])$ others ud	$out[3][3] := 1 - 1/\text{sqrt}(kq[3][3])$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$kq[2][2] := 1 - 1/kq[2][2]$ others ud	$out[2][2] := 1 - kq[2][2]$ others ud	$out[2][2] := kq[2][2]/(1 + kq[2][2])$ others ud.	kq	$out[2][2] := 1 - \text{sqrt}(kq[2][2])$ others ud	$out[2][2] := 1 - 1/\text{sqrt}(kq[2][2])$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := 1 - 1/kq[2][2]$ $out[3][3] := 1 - 1/kq[3][3]$ others ud	$out[2][2] := 1 - kq[2][2]$ $out[3][3] := 1 - kq[3][3]$ others ud	$out[2][2] := kq[2][2]/(1 + kq[2][2])$ $out[3][3] := kq[3][3]/(1 + kq[3][3])$ others ud.	kq	$out[2][2] := 1 - \text{sqrt}(kq[2][2])$ $out[3][3] := 1 - \text{sqrt}(kq[3][3])$ others ud	$out[2][2] := 1 - 1/\text{sqrt}(kq[2][2])$ $out[3][3] := 1 - 1/\text{sqrt}(kq[3][3])$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := 1 - 1/kq[1][1]$ others ud	$out[1][1] := 1 - kq[1][1]$ others ud	$out[1][1] := kq[1][1]/(1 + kq[1][1])$ others ud.	kq	$out[1][1] := 1 - \text{sqrt}(kq[1][1])$ others ud	$out[1][1] := 1 - 1/\text{sqrt}(kq[1][1])$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := 1 - 1/kq[1][1]$ $out[3][3] := 1 - 1/kq[3][3]$ others ud	$out[1][1] := 1 - kq[1][1]$ $out[3][3] := 1 - kq[3][3]$ others ud	$out[1][1] := kq[1][1]/(1 + kq[1][1])$ $out[3][3] := kq[3][3]/(1 + kq[3][3])$ others ud.	kq	$out[1][1] := 1 - \text{sqrt}(kq[1][1])$ $out[3][3] := 1 - \text{sqrt}(kq[3][3])$ others ud	$out[1][1] := 1 - 1/\text{sqrt}(kq[1][1])$ $out[3][3] := 1 - 1/\text{sqrt}(kq[3][3])$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := 1 - 1/kq[1][1]$ $out[2][2] := 1 - 1/kq[2][2]$ others ud	$out[1][1] := 1 - kq[1][1]$ $out[2][2] := 1 - kq[2][2]$ others ud	$out[1][1] := kq[1][1]/(1 + kq[1][1])$ $out[2][2] := kq[2][2]/(1 + kq[2][2])$ others ud.	kq	$out[1][1] := 1 - \text{sqrt}(kq[1][1])$ $out[2][2] := 1 - \text{sqrt}(kq[2][2])$ others ud	$out[1][1] := 1 - 1/\text{sqrt}(kq[1][1])$ $out[2][2] := 1 - 1/\text{sqrt}(kq[2][2])$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := 1 - 1/kq[1][1]$ $out[2][2] := 1 - 1/kq[2][2]$ $out[3][3] := 1 - 1/kq[3][3]$ others ud	$out[1][1] := 1 - kq[1][1]$ $out[2][2] := 1 - kq[2][2]$ $out[3][3] := 1 - kq[3][3]$ others ud	$out[1][1] := kq[1][1]/(1 + kq[1][1])$ $out[2][2] := kq[2][2]/(1 + kq[2][2])$ $out[3][3] := kq[3][3]/(1 + kq[3][3])$ others ud.	kq	$out[1][1] := 1 - \text{sqrt}(kq[1][1])$ $out[2][2] := 1 - \text{sqrt}(kq[2][2])$ $out[3][3] := 1 - \text{sqrt}(kq[3][3])$ others ud	$out[1][1] := 1 - 1/\text{sqrt}(kq[1][1])$ $out[2][2] := 1 - 1/\text{sqrt}(kq[2][2])$ $out[3][3] := 1 - 1/\text{sqrt}(kq[3][3])$ others ud.

G

Table D-15 out for sdpg_g_knownquantity

$H_1 \wedge H_2 \rightarrow G$

Continue→

$H_2 : \text{kgflag}$

H_1

	lfst	efst	fist	eist	tst
	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$	kq	$\text{Out} := [\text{ud}]$
$kq[1][1] = \text{ud} \wedge kq[2][2] = \text{ud} \wedge kq[3][3] = \text{ud}$	$\text{out}[3][3] := 1 - 1/\sqrt{2 * kq[3][3] + 1}$ others ud.	$\text{out}[3][3] := 1 - \sqrt{1 - 2 * kq[3][3]}$ others ud	$\text{out}[3][3] := kq[3][3]/(1 + kq[3][3])$ others ud.	kq	$\text{out}[3][3] := 1 - \exp(-kq[3][3])$ others ud.
$kq[1][1] = \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] = \text{ud}$	$\text{out}[2][2] := 1 - 1/\sqrt{2 * kq[2][2] + 1}$ others ud.	$\text{out}[2][2] := 1 - \sqrt{1 - 2 * kq[2][2]}$ others ud	$\text{out}[2][2] := kq[2][2]/(1 + kq[2][2])$ others ud.	kq	$\text{out}[2][2] := 1 - \exp(-kq[2][2])$ others ud.
$kq[1][1] = \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[2][2] := 1 - 1/\sqrt{2 * kq[2][2] + 1}$ $\text{out}[3][3] := 1 - 1/\sqrt{2 * kq[3][3] + 1}$ others ud.	$\text{out}[2][2] := 1 - \sqrt{1 - 2 * kq[2][2]}$ $\text{out}[3][3] := 1 - \sqrt{1 - 2 * kq[3][3]}$ others ud	$\text{out}[2][2] := kq[2][2]/(1 + kq[2][2])$ $\text{out}[3][3] := kq[3][3]/(1 + kq[3][3])$ others ud.	kq	$\text{out}[2][2] := 1 - \exp(-kq[2][2])$ $\text{out}[3][3] := 1 - \exp(-kq[3][3])$ others ud.
$kq[1][1] \neq \text{ud} \wedge kq[2][2] = \text{ud} \wedge kq[3][3] = \text{ud}$	$\text{out}[1][1] := 1 - 1/\sqrt{2 * kq[1][1] + 1}$ others ud.	$\text{out}[1][1] := 1 - \sqrt{1 - 2 * kq[1][1]}$ others ud	$\text{out}[1][1] := kq[1][1]/(1 + kq[1][1])$ others ud.	kq	$\text{out}[1][1] := 1 - \exp(-kq[1][1])$ others ud.
$kq[1][1] \neq \text{ud} \wedge kq[2][2] = \text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[1][1] := 1 - 1/\sqrt{2 * kq[1][1] + 1}$ $\text{out}[3][3] := 1 - 1/\sqrt{2 * kq[3][3] + 1}$ others ud.	$\text{out}[1][1] := 1 - \sqrt{1 - 2 * kq[1][1]}$ $\text{out}[3][3] := 1 - \sqrt{1 - 2 * kq[3][3]}$ others ud	$\text{out}[1][1] := kq[1][1]/(1 + kq[1][1])$ $\text{out}[3][3] := kq[3][3]/(1 + kq[3][3])$ others ud.	kq	$\text{out}[1][1] := 1 - \exp(-kq[1][1])$ $\text{out}[3][3] := 1 - \exp(-kq[3][3])$ others ud.
$kq[1][1] \neq \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] = \text{ud}$	$\text{out}[1][1] := 1 - 1/\sqrt{2 * kq[1][1] + 1}$ $\text{out}[2][2] := 1 - 1/\sqrt{2 * kq[2][2] + 1}$ others ud.	$\text{out}[1][1] := 1 - \sqrt{1 - 2 * kq[1][1]}$ $\text{out}[2][2] := 1 - \sqrt{1 - 2 * kq[2][2]}$ others ud	$\text{out}[1][1] := kq[1][1]/(1 + kq[1][1])$ $\text{out}[2][2] := kq[2][2]/(1 + kq[2][2])$ others ud.	kq	$\text{out}[1][1] := 1 - \exp(-kq[1][1])$ $\text{out}[2][2] := 1 - \exp(-kq[2][2])$ others ud
$kq[1][1] \neq \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[1][1] := 1 - 1/\sqrt{2 * kq[1][1] + 1}$ $\text{out}[2][2] := 1 - 1/\sqrt{2 * kq[2][2] + 1}$ $\text{out}[3][3] := 1 - 1/\sqrt{2 * kq[3][3] + 1}$ others ud.	$\text{out}[1][1] := 1 - \sqrt{1 - 2 * kq[1][1]}$ $\text{out}[2][2] := 1 - \sqrt{1 - 2 * kq[2][2]}$ $\text{out}[3][3] := 1 - \sqrt{1 - 2 * kq[3][3]}$ others ud	$\text{out}[1][1] := kq[1][1]/(1 + kq[1][1])$ $\text{out}[2][2] := kq[2][2]/(1 + kq[2][2])$ $\text{out}[3][3] := kq[3][3]/(1 + kq[3][3])$ others ud.	kq	$\text{out}[1][1] := 1 - \exp(-kq[1][1])$ $\text{out}[2][2] := 1 - \exp(-kq[2][2])$ $\text{out}[3][3] := 1 - \exp(-kq[3][3])$ others ud

→Continue

G

Table D-15 out for sdpg_g_knownquantity (Continue)

$H_1 \wedge H_2 \rightarrow G$

H	$\text{disp.dispu}=\text{ud}\wedge$ $\text{disp.dispv}=\text{ud}\wedge$ $\text{disp.dispw}=\text{ud}$	Out :=[ud]
	$\text{disp.dispu}=\text{ud}\wedge$ $\text{disp.dispv}=\text{ud}\wedge$ $\text{disp.dispw}\neq\text{ud}$	$\text{out}[3][3] := \text{disp.dispw} / (\text{disp.dispw} + \text{sg.width})$ others ud
	$\text{disp.dispu}=\text{ud}\wedge$ $\text{disp.dispv}\neq\text{ud}\wedge$ $\text{disp.dispw}=\text{ud}$	$\text{out}[2][2] := \text{disp.dispv} / (\text{disp.v} + \text{sg.height})$ others ud
	$\text{disp.dispu}=\text{ud}\wedge$ $\text{disp.dispv}\neq\text{ud}\wedge$ $\text{disp.dispw}\neq\text{ud}$	$\text{out}[2][2] := \text{disp.dispv} / (\text{disp.dispv} + \text{sg.height})$ $\text{out}[3][3] := \text{disp.dispw} / (\text{disp.dispw} + \text{sg.width})$ others ud
	$\text{disp.dispu}\neq\text{ud}\wedge$ $\text{disp.dispv}=\text{ud}\wedge$ $\text{disp.dispw}=\text{ud}$	$\text{out}[1][1] := \text{disp.dispu} / (\text{disp.dispu} + \text{sg.length})$ others ud
	$\text{disp.dispu}\neq\text{ud}\wedge$ $\text{disp.dispv}=\text{ud}\wedge$ $\text{disp.dispw}\neq\text{ud}$	$\text{out}[1][1] := \text{disp.dispu} / (\text{disp.dispu} + \text{sg.length})$ $\text{out}[3][3] := \text{disp.dispw} / (\text{disp.dispw} + \text{sg.width})$ others ud
	$\text{disp.dispu}\neq\text{ud}\wedge$ $\text{disp.dispv}\neq\text{ud}\wedge$ $\text{disp.dispw}=\text{ud}$	$\text{out}[1][1] := \text{disp.dispu} / (\text{disp.dispu} + \text{sg.length})$ $\text{out}[2][2] := \text{disp.dispv} / (\text{disp.dispv} + \text{sg.height})$ others ud
	$\text{disp.dispu}\neq\text{ud}\wedge$ $\text{disp.dispv}\neq\text{ud}\wedge$ $\text{disp.dispw}\neq\text{ud}$	$\text{out}[1][1] := \text{disp.dispu} / (\text{disp.dispu} + \text{sg.length})$ $\text{out}[2][2] := \text{disp.dispv} / (\text{disp.dispv} + \text{sg.height})$ $\text{out}[3][3] := \text{disp.dispw} / (\text{disp.dispw} + \text{sg.width})$ others ud

Table D-16 out for sdpg_g_geometry

D. 3.19 Cauchy deformation tensor module

Prefix: cdt_

Reference: MG C. 3.2.13

D. 3.19.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

TensorFlagT from the tensor data definition module

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
cdt_g_knownquantity	TensorDataT TensorFlagT	TensorDataT	
cdt_g_geometry	SpecimenGeometryT DisplacementT	TensorDataT	none

D. 3.19.2 Interface semantics

State variable: none

Local variable:

temp: Real*[DIM][DIM]

State invariant: none

Assumption:

Access routine semantics

cdt_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)

exception: Table D-17

output: Table D-18

cdt_g_geometry(sg: SpecimenGeometryT, disp: DisplacementT)

exception: none

output: Table 4.4.19-3

H_1	mdg	sdg	mdpg	sdpg	cdt	gdt	lfst	efst	list	eist	tst
G	(kq[1][1]=0) \vee (kq[2][2]=0) \vee (kq[3][3]=0) \Rightarrow d_zero	non	(kq[1][1]=-1) \vee (kq[2][2]=-1) \vee (kq[3][3]=-1) \Rightarrow d_zero	non	non	(kq[1][1]=0) \vee (kq[2][2]=0) \vee (kq[3][3]=0) \Rightarrow d_zero	(kq[1][1]=-0.5) \vee (kq[2][2]=-0.5) \vee (kq[3][3]=-0.5) \Rightarrow d_zero	non	(kq[1][1]=-1) \vee (kq[2][2]=-1) \vee (kq[3][3]=-1) \Rightarrow d_zero	non	non

$H_1 \rightarrow G$

Table D-17 exception for cdt_g_knownquantity

H₁

H₂ : kgflag

	mdg	sdg	mdpg	sdpg	cdt	gdt
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]=ud	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	kq	Out:=[ud]
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[3][3] := (kq[3][3]) ⁻² others ud	out[3][3] := (kq[3][3]) ² others ud	out[3][3] := (1 + kq[3][3]) ⁻² others ud.	out[3][3] := (1 - kq[3][3]) ⁻² others ud.	kq	out[3][3] := 1 / kq[3][3] others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[2][2] := (kq[2][2]) ⁻² others ud	out[2][2] := (kq[2][2]) ² others ud	out[2][2] := (1 + kq[2][2]) ⁻² others ud.	out[2][2] := (1 - kq[2][2]) ⁻² others ud.	kq	out[2][2] := 1 / kq[2][2] others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[2][2] := (kq[2][2]) ⁻² out[3][3] := (kq[3][3]) ⁻² others ud	out[2][2] := (kq[2][2]) ² out[3][3] := (kq[3][3]) ² others ud	out[2][2] := (1 + kq[2][2]) ⁻² out[3][3] := (1 + kq[3][3]) ⁻² others ud.	out[2][2] := (1 - kq[2][2]) ⁻² out[3][3] := (1 - kq[3][3]) ⁻² others ud.	kq	out[2][2] := 1 / kq[2][2] out[3][3] := 1 / kq[3][3] others ud.
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]=ud	out[1][1] := (kq[1][1]) ⁻² others ud	out[1][1] := (kq[1][1]) ² others ud	out[1][1] := (1 + kq[1][1]) ⁻² others ud.	out[1][1] := (1 - kq[1][1]) ⁻² others ud.	kq	out[1][1] := 1 / kq[1][1] others ud.
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[1][1] := (kq[1][1]) ⁻² out[3][3] := (kq[3][3]) ⁻² others ud	out[1][1] := (kq[1][1]) ² out[3][3] := (kq[3][3]) ² others ud	out[1][1] := (1 + kq[1][1]) ⁻² out[3][3] := (1 + kq[3][3]) ⁻² others ud.	out[1][1] := (1 - kq[1][1]) ⁻² out[3][3] := (1 - kq[3][3]) ⁻² others ud.	kq	out[1][1] := 1 / kq[1][1] out[3][3] := 1 / kq[3][3] others ud
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[1][1] := (kq[1][1]) ⁻² out[2][2] := (kq[2][2]) ⁻² others ud	out[1][1] := (kq[1][1]) ² out[2][2] := (kq[2][2]) ² others ud	out[1][1] := (1 + kq[1][1]) ⁻² out[2][2] := (1 + kq[2][2]) ⁻² others ud	out[1][1] := (1 - kq[1][1]) ⁻² out[2][2] := (1 - kq[2][2]) ⁻² others ud	kq	out[1][1] := 1 / kq[1][1] out[2][2] := 1 / kq[2][2] others ud
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[1][1] := (kq[1][1]) ⁻² out[2][2] := (kq[2][2]) ⁻² out[3][3] := (kq[3][3]) ⁻² others ud	out[1][1] := (kq[1][1]) ² out[2][2] := (kq[2][2]) ² out[3][3] := (kq[3][3]) ² others ud	out[1][1] := (1 + kq[1][1]) ⁻² out[2][2] := (1 + kq[2][2]) ⁻² out[3][3] := (1 + kq[3][3]) ⁻² others ud.	out[1][1] := (1 - kq[1][1]) ⁻² out[2][2] := (1 - kq[2][2]) ⁻² out[3][3] := (1 - kq[3][3]) ⁻² others ud	Kq	out[1][1] := 1 / kq[1][1] out[2][2] := 1 / kq[2][2] out[3][3] := 1 / kq[3][3] others ud

G Table D-18 out for cdt_g_knownquantity

H₁ ^ H₂ → G

Continue→

H₁

	lfst	efst	list	eist	tst
	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]=ud					
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[3][3] := 1/(2 * kq[3][3] + 1) others ud.	out[3][3] := (1 - 2 * kq[3][3]) others ud.	out[3][3] := 1/(1 + kq[3][3]) ² others ud.	out[3][3] := (1 - kq[3][3]) ² others ud.	out[3][3] := exp(-2 * kq[3][3]) others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[2][2] := 1/(2 * kq[2][2] + 1) others ud.	out[2][2] := (1 - 2 * kq[2][2]) others ud.	out[2][2] := 1/(1 + kq[2][2]) ² others ud.	out[2][2] := (1 - kq[2][2]) ² others ud.	out[2][2] := exp(-2 * kq[2][2]) others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[2][2] := 1/(2 * kq[2][2] + 1) out[3][3] := 1/(2 * kq[3][3] + 1) others ud.	out[2][2] := (1 - 2 * kq[2][2]) out[3][3] := (1 - 2 * kq[3][3]) others ud.	out[2][2] := 1/(1 + kq[2][2]) ² out[3][3] := 1/(1 + kq[3][3]) ² others ud.	out[2][2] := (1 - kq[2][2]) ² out[3][3] := (1 - kq[3][3]) ² others ud.	out[2][2] := exp(-2 * kq[2][2]) out[3][3] := exp(-2 * kq[3][3]) others ud.
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]=ud	out[1][1] := 1/(2 * kq[1][1] + 1) others ud.	out[1][1] := (1 - 2 * kq[1][1]) others ud.	out[1][1] := 1/(1 + kq[1][1]) ² others ud.	out[1][1] := (1 - kq[1][1]) ² others ud.	out[1][1] := exp(-2 * kq[1][1]) others ud.
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[1][1] := 1/(2 * kq[1][1] + 1) out[3][3] := 1/(2 * kq[3][3] + 1) others ud.	out[1][1] := (1 - 2 * kq[1][1]) out[3][3] := (1 - 2 * kq[3][3]) others ud.	out[1][1] := 1/(1 + kq[1][1]) ² out[3][3] := 1/(1 + kq[3][3]) ² others ud.	out[1][1] := (1 - kq[1][1]) ² out[3][3] := (1 - kq[3][3]) ² others ud.	out[1][1] := exp(-2 * kq[1][1]) out[3][3] := exp(-2 * kq[3][3]) others ud.
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[1][1] := 1/(2 * kq[1][1] + 1) out[2][2] := 1/(2 * kq[2][2] + 1) others ud.	out[1][1] := (1 - 2 * kq[1][1]) out[2][2] := (1 - 2 * kq[2][2]) others ud.	out[1][1] := 1/(1 + kq[1][1]) ² out[2][2] := 1/(1 + kq[2][2]) ² others ud.	out[1][1] := (1 - kq[1][1]) ² out[2][2] := (1 - kq[2][2]) ² others ud.	out[1][1] := exp(-2 * kq[1][1]) out[2][2] := exp(-2 * kq[2][2]) others ud.
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[1][1] := 1/(2 * kq[1][1] + 1) out[2][2] := 1/(2 * kq[2][2] + 1) out[3][3] := 1/(2 * kq[3][3] + 1) others ud.	out[1][1] := (1 - 2 * kq[1][1]) out[2][2] := (1 - 2 * kq[2][2]) out[3][3] := (1 - 2 * kq[3][3]) others ud.	out[1][1] := 1/(1 + kq[1][1]) ² out[2][2] := 1/(1 + kq[2][2]) ² out[3][3] := 1/(1 + kq[3][3]) ² others ud.	out[1][1] := (1 - kq[1][1]) ² out[2][2] := (1 - kq[2][2]) ² out[3][3] := (1 - kq[3][3]) ² others ud.	out[1][1] := exp(-2 * kq[1][1]) out[2][2] := exp(-2 * kq[2][2]) out[3][3] := exp(-2 * kq[3][3]) others ud.

→Continue

G

Table D-18 out for cdt_g_knownquantity (Continue)

H₁ ∧ H₂ → G

G

H→G

H	$\text{disp.dispu}=\text{ud} \wedge$ $\text{disp.dispv}=\text{ud} \wedge$ $\text{disp.dispw}=\text{ud}$	Out :=[ud]
	$\text{disp.dispu}=\text{ud} \wedge$ $\text{disp.dispv}=\text{ud} \wedge$ $\text{disp.dispw} \neq \text{ud}$	$\text{out}[3][3] := (1 + \text{disp.dispw} / \text{sg.width})^{-2}$ others ud
	$\text{disp.dispu}=\text{ud} \wedge$ $\text{disp.dispv} \neq \text{ud} \wedge$ $\text{disp.dispw}=\text{ud}$	$\text{out}[2][2] := (1 + \text{disp.dispv} / \text{sg.height})^{-2}$ others ud
	$\text{disp.dispu}=\text{ud} \wedge$ $\text{disp.dispv} \neq \text{ud} \wedge$ $\text{disp.dispw} \neq \text{ud}$	$\text{out}[2][2] := (1 + \text{disp.dispv} / \text{sg.height})^{-2}$ $\text{out}[3][3] := (1 + \text{disp.dispw} / \text{sg.width})^{-2}$ others ud
	$\text{disp.dispu} \neq \text{ud} \wedge$ $\text{disp.dispv}=\text{ud} \wedge$ $\text{disp.dispw}=\text{ud}$	$\text{out}[1][1] := (1 + \text{disp.dispu} / \text{sg.length})^{-2}$ others ud
	$\text{disp.dispu} \neq \text{ud} \wedge$ $\text{disp.dispv}=\text{ud} \wedge$ $\text{disp.dispw} \neq \text{ud}$	$\text{out}[1][1] := (1 + \text{disp.dispu} / \text{sg.length})^{-2}$ $\text{out}[3][3] := (1 + \text{disp.dispw} / \text{sg.width})^{-2}$ others ud
	$\text{disp.dispu} \neq \text{ud} \wedge$ $\text{disp.dispv} \neq \text{ud} \wedge$ $\text{disp.dispw}=\text{ud}$	$\text{out}[1][1] := (1 + \text{disp.dispu} / \text{sg.length})^{-2}$ $\text{out}[2][2] := (1 + \text{disp.dispv} / \text{sg.height})^{-2}$ others ud
	$\text{disp.dispu} \neq \text{ud} \wedge$ $\text{disp.dispv} \neq \text{ud} \wedge$ $\text{disp.dispw} \neq \text{ud}$	$\text{out}[1][1] := (1 + \text{disp.dispu} / \text{sg.length})^{-2}$ $\text{out}[2][2] := (1 + \text{disp.dispv} / \text{sg.height})^{-2}$ $\text{out}[3][3] := (1 + \text{disp.dispw} / \text{sg.width})^{-2}$ others ud

Table D-19 out for cdt_g_geometry

D. 3.20 Green deformation tensor module

Prefix: gdt_

Reference: MG C. 3.2.13

D. 3.20.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

TensorFlagT from the tensor data definition module

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
gdt_g_knownquantity	TensorDataT TensorFlagT	TensorDataT	
gdt_g_geometry	SpecimenGeometryT DisplacementT	TensorDataT	None

D. 3.20.2 Interface semantics

State variable: none

State invariant: none

Assumption:

Access routine semantics

cdt_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)

exception: Table D-20

output: Table D-21

cdt_g_geometry(sg: SpecimenGeometryT, disp: DisplacementT)

exception: none

Output: Table D-22

H_1	mdg	sdg	mdpg	sdpg	cdt	gdt	lfst	efst	list	eist	tst
G	non	(kq[1][1]=0) \vee (kq[2][2]=0) \vee (kq[3][3]=0) \Rightarrow d_zero	non	(kq[1][1]=-1) \vee (kq[2][2]=-1) \vee (kq[3][3]=-1) \Rightarrow d_zero	(kq[1][1]=0) \vee (kq[2][2]=0) \vee (kq[3][3]=0) \Rightarrow d_zero	non	non	(kq[1][1]=-0.5) \vee (kq[2][2]=-0.5) \vee (kq[3][3]=-0.5) \Rightarrow d_zero	non	(kq[1][1]=-1) \vee (kq[2][2]=-1) \vee (kq[3][3]=-1) \Rightarrow d_zero	non

$H_1 \rightarrow G$

Table D-20 exception for gdt_g_knownquantity

H₁

H₂ : kgflag

	mdg	sdg	mdpg	sdpg	cdt	gdt
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	kq
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := (kq[3][3])^2$ others ud	$out[3][3] := (kq[3][3])^{-2}$ others ud	$out[3][3] := (1 + kq[3][3])^2$ others ud.	$out[3][3] := (1 - kq[3][3])^{-2}$ others ud.	$out[3][3] := 1 / kq[3][3]$ others ud.	kq
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[2][2] := (kq[2][2])^2$ others ud	$out[2][2] := (kq[2][2])^{-2}$ others ud	$out[2][2] := (1 + kq[2][2])^2$ others ud.	$out[2][2] := (1 - kq[2][2])^{-2}$ others ud.	$out[2][2] := 1 / kq[2][2]$ others ud.	kq
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := (kq[2][2])^2$ $out[3][3] := (kq[3][3])^2$ others ud	$out[2][2] := (kq[2][2])^{-2}$ $out[3][3] := (kq[3][3])^{-2}$ others ud	$out[2][2] := (1 + kq[2][2])^2$ $out[3][3] := (1 + kq[3][3])^2$ others ud.	$out[2][2] := (1 - kq[2][2])^{-2}$ $out[3][3] := (1 - kq[3][3])^{-2}$ others ud.	$out[2][2] := 1 / kq[2][2]$ $out[3][3] := 1 / kq[3][3]$ others ud.	kq
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := (kq[1][1])^2$ others ud	$out[1][1] := (kq[1][1])^{-2}$ others ud	$out[1][1] := (1 + kq[1][1])^2$ others ud.	$out[1][1] := (1 - kq[1][1])^{-2}$ others ud.	$out[1][1] := 1 / kq[1][1]$ others ud.	kq
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := (kq[1][1])^2$ $out[3][3] := (kq[3][3])^2$ others ud	$out[1][1] := (kq[1][1])^{-2}$ $out[3][3] := (kq[3][3])^{-2}$ others ud	$out[1][1] := (1 + kq[1][1])^2$ $out[3][3] := (1 + kq[3][3])^2$ others ud.	$out[1][1] := (1 - kq[1][1])^{-2}$ $out[3][3] := (1 - kq[3][3])^{-2}$ others ud.	$out[1][1] := 1 / kq[1][1]$ $out[3][3] := 1 / kq[3][3]$ others ud	kq
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := (kq[1][1])^2$ $out[2][2] := (kq[2][2])^2$ others ud	$out[1][1] := (kq[1][1])^{-2}$ $out[2][2] := (kq[2][2])^{-2}$ others ud	$out[1][1] := (1 + kq[1][1])^2$ $out[2][2] := (1 + kq[2][2])^2$ others ud	$out[1][1] := (1 - kq[1][1])^{-2}$ $out[2][2] := (1 - kq[2][2])^{-2}$ others ud	$out[1][1] := 1 / kq[1][1]$ $out[2][2] := 1 / kq[2][2]$ others ud	kq
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := (kq[1][1])^2$ $out[2][2] := (kq[2][2])^2$ $out[3][3] := (kq[3][3])^2$ others ud	$out[1][1] := (kq[1][1])^{-2}$ $out[2][2] := (kq[2][2])^{-2}$ $out[3][3] := (kq[3][3])^{-2}$ others ud	$out[1][1] := (1 + kq[1][1])^2$ $out[2][2] := (1 + kq[2][2])^2$ $out[3][3] := (1 + kq[3][3])^2$ others ud.	$out[1][1] := (1 - kq[1][1])^{-2}$ $out[2][2] := (1 - kq[2][2])^{-2}$ $out[3][3] := (1 - kq[3][3])^{-2}$ others ud	$out[1][1] := 1 / kq[1][1]$ $out[2][2] := 1 / kq[2][2]$ $out[3][3] := 1 / kq[3][3]$ others ud	Kq

G Table D-21 out for gdt_g_knownquantity

H₁ ∧ H₂ → G

Continue→

$H_2 : \text{kgflag}$

H_1

	lfst	efst	list	eist	tst
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := (2 * kq[3][3] + 1)$ others ud.	$out[3][3] := 1 / (1 - 2 * kq[3][3])$ others ud.	$out[3][3] := (1 + kq[3][3])^2$ others ud.	$out[3][3] := (1 - kq[3][3])^{-2}$ others ud.	$out[3][3] := \exp(2 * kq[3][3])$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[2][2] := (2 * kq[2][2] + 1)$ others ud.	$out[2][2] := 1 / (1 - 2 * kq[2][2])$ others ud.	$out[2][2] := (1 + kq[2][2])^2$ others ud.	$out[2][2] := (1 - kq[2][2])^{-2}$ others ud.	$out[2][2] := \exp(2 * kq[2][2])$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := (2 * kq[2][2] + 1)$ $out[3][3] := (2 * kq[3][3] + 1)$ others ud.	$out[2][2] := 1 / (1 - 2 * kq[2][2])$ $out[3][3] := 1 / (1 - 2 * kq[3][3])$ others ud.	$out[2][2] := (1 + kq[2][2])^2$ $out[3][3] := (1 + kq[3][3])^2$ others ud.	$out[2][2] := (1 - kq[2][2])^{-2}$ $out[3][3] := (1 - kq[3][3])^{-2}$ others ud.	$out[2][2] := \exp(2 * kq[2][2])$ $out[3][3] := \exp(2 * kq[3][3])$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := (2 * kq[1][1] + 1)$ others ud.	$out[1][1] := 1 / (1 - 2 * kq[1][1])$ others ud.	$out[1][1] := (1 + kq[1][1])^2$ others ud.	$out[1][1] := (1 - kq[1][1])^{-2}$ others ud.	$out[1][1] := \exp(2 * kq[1][1])$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := (2 * kq[1][1] + 1)$ $out[3][3] := (2 * kq[3][3] + 1)$ others ud.	$out[1][1] := 1 / (1 - 2 * kq[1][1])$ $out[3][3] := 1 / (1 - 2 * kq[3][3])$ others ud.	$out[1][1] := (1 + kq[1][1])^2$ $out[3][3] := (1 + kq[3][3])^2$ others ud.	$out[1][1] := (1 - kq[1][1])^{-2}$ $out[3][3] := (1 - kq[3][3])^{-2}$ others ud.	$out[1][1] := \exp(2 * kq[1][1])$ $out[3][3] := \exp(2 * kq[3][3])$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := (2 * kq[1][1] + 1)$ $out[2][2] := (2 * kq[2][2] + 1)$ others ud.	$out[1][1] := 1 / (1 - 2 * kq[1][1])$ $out[2][2] := 1 / (1 - 2 * kq[2][2])$ others ud.	$out[1][1] := (1 + kq[1][1])^2$ $out[2][2] := (1 + kq[2][2])^2$ others ud.	$out[1][1] := (1 - kq[1][1])^{-2}$ $out[2][2] := (1 - kq[2][2])^{-2}$ others ud.	$out[1][1] := \exp(2 * kq[1][1])$ $out[2][2] := \exp(2 * kq[2][2])$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := (2 * kq[1][1] + 1)$ $out[2][2] := (2 * kq[2][2] + 1)$ $out[3][3] := (2 * kq[3][3] + 1)$ others ud.	$out[1][1] := 1 / (1 - 2 * kq[1][1])$ $out[2][2] := 1 / (1 - 2 * kq[2][2])$ $out[3][3] := 1 / (1 - 2 * kq[3][3])$ others ud.	$out[1][1] := (1 + kq[1][1])^2$ $out[2][2] := (1 + kq[2][2])^2$ $out[3][3] := (1 + kq[3][3])^2$ others ud.	$out[1][1] := (1 - kq[1][1])^{-2}$ $out[2][2] := (1 - kq[2][2])^{-2}$ $out[3][3] := (1 - kq[3][3])^{-2}$ others ud.	$out[1][1] := \exp(2 * kq[1][1])$ $out[2][2] := \exp(2 * kq[2][2])$ $out[3][3] := \exp(2 * kq[3][3])$ others ud.

→Continue

G

Table D-21 out for gdt_g_knownquantity (Continue)

$H_1 \wedge H_2 \rightarrow G$

G

H→G

H

disp.dispu=ud^ disp.dispv=ud^ disp.dispw=ud	Out :=[ud]
disp.dispu=ud^ disp.dispv=ud^ disp.dispw≠ud	out[3][3] := (1 + disp.dispw / sg.width) ² others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw=ud	out[2][2] := (1 + disp.dispv / sg.height) ² others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw≠ud	out[2][2] := (1 + disp.dispv / sg.height) ² out[3][3] := (1 + disp.dispw / sg.width) ² others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw=ud	out[1][1] := (1 + disp.dispu / sg.length) ⁻² others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw≠ud	out[1][1] := (1 + disp.dispu / sg.length) ² out[3][3] := (1 + disp.dispw / sg.width) ² others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw=ud	out[1][1] := (1 + disp.dispu / sg.length) ² out[2][2] := (1 + disp.dispv / sg.height) ² others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw≠ud	out[1][1] := (1 + disp.dispu / sg.length) ² out[2][2] := (1 + disp.dispv / sg.height) ² out[3][3] := (1 + disp.dispw / sg.width) ² others ud

Table D-22 out for gdt_g_geometry

D. 3.21 Lagrangian (Green's) finite strain tensor module

Prefix: `lfst_`

Reference: MG – C. 3.2.20

D. 3.21.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

TensorFlagT from the tensor data definition module

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
<code>lfst_g_knownquantity</code>	TensorDataT TensorFlagT	TensorDataT	
<code>lfst_g_geometry</code>	SpecimenGeometryT DisplacementT	TensorDataT	none

D. 3.21.2 Interface semantics

State variable: none

State invariant: none

Assumption:

Access routine semantics

`lfst_g_knownquantity`(kq: TensorDataT, kqflag: TensorFlagT)

Exception: Table D-23

Output: Table D-24

`lfst_g_geometry`(sg: SpecimenGeometryT, disp: DisplacementT)

Exception: None

Output: Table D-25

H_1	mdg	sdg	mdpg	sdpd	cdt	gdt	lfst	efst	list	eist	tst
G	non	(kq[1][1]=0) \vee (kq[2][2]=0) \vee (kq[3][3]=0) \Rightarrow d_zero	non	(kq[1][1]=1) \vee (kq[2][2]=1) \vee (kq[3][3]=1) \Rightarrow d_zero	(kq[1][1]=0) \vee (kq[2][2]=0) \vee (kq[3][3]=0) \Rightarrow d_zero	non	non	(kq[1][1]=0.5) \vee (kq[2][2]=0.5) \vee (kq[3][3]=0.5) \Rightarrow d_zero	non	(kq[1][1]=1) \vee (kq[2][2]=1) \vee (kq[3][3]=1) \Rightarrow d_zero	non

$H_1 \rightarrow G$

Table D-23 exception for lfst_g_knownquantity

$H_2 : \text{kgflag}$

H_1

	mdg	sdg	mdpg	sdpg
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := ((kq[3][3])^2 - 1) / 2$ others ud	$out[3][3] := ((kq[3][3])^{-2} - 1) / 2$ others ud	$out[3][3] := ((1 + kq[3][3])^2 - 1) / 2$ others ud.	$out[3][3] := ((kq[3][3] / (kq[3][3] - 1))^2 - 1) / 2$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[2][2] := ((kq[2][2])^2 - 1) / 2$ others ud	$out[2][2] := ((kq[2][2])^{-2} - 1) / 2$ others ud	$out[2][2] := ((1 + kq[2][2])^2 - 1) / 2$ others ud.	$out[2][2] := ((kq[2][2] / (kq[2][2] - 1))^2 - 1) / 2$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := ((kq[2][2])^2 - 1) / 2$ $out[3][3] := ((kq[3][3])^2 - 1) / 2$ others ud	$out[2][2] := ((kq[2][2])^{-2} - 1) / 2$ $out[3][3] := ((kq[3][3])^{-2} - 1) / 2$ others ud	$out[2][2] := ((1 + kq[2][2])^2 - 1) / 2$ $out[3][3] := ((1 + kq[3][3])^2 - 1) / 2$ others ud	$out[2][2] := ((kq[2][2] / (kq[2][2] - 1))^2 - 1) / 2$ $out[3][3] := ((kq[3][3] / (kq[3][3] - 1))^2 - 1) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := ((kq[1][1])^2 - 1) / 2$ others ud	$out[1][1] := ((kq[1][1])^{-2} - 1) / 2$ others ud	$out[1][1] := ((1 + kq[1][1])^2 - 1) / 2$ others ud.	$out[1][1] := ((kq[1][1] / (kq[1][1] - 1))^2 - 1) / 2$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := ((kq[1][1])^2 - 1) / 2$ $out[3][3] := ((kq[3][3])^2 - 1) / 2$ others ud	$out[1][1] := ((kq[1][1])^{-2} - 1) / 2$ $out[3][3] := ((kq[3][3])^{-2} - 1) / 2$ others ud	$out[1][1] := ((1 + kq[1][1])^2 - 1) / 2$ $out[3][3] := ((1 + kq[3][3])^2 - 1) / 2$ others ud	$out[1][1] := ((kq[1][1] / (kq[1][1] - 1))^2 - 1) / 2$ $out[3][3] := ((kq[3][3] / (kq[3][3] - 1))^2 - 1) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := ((kq[1][1])^2 - 1) / 2$ $out[2][2] := ((kq[2][2])^2 - 1) / 2$ others ud	$out[1][1] := ((kq[1][1])^{-2} - 1) / 2$ $out[2][2] := ((kq[2][2])^{-2} - 1) / 2$ others ud	$out[1][1] := ((1 + kq[1][1])^2 - 1) / 2$ $out[2][2] := ((1 + kq[2][2])^2 - 1) / 2$ others ud	$out[1][1] := ((kq[1][1] / (kq[1][1] - 1))^2 - 1) / 2$ $out[2][2] := ((kq[2][2] / (kq[2][2] - 1))^2 - 1) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := ((kq[1][1])^2 - 1) / 2$ $out[2][2] := ((kq[2][2])^2 - 1) / 2$ $out[3][3] := ((kq[3][3])^2 - 1) / 2$ others ud	$out[1][1] := ((kq[1][1])^{-2} - 1) / 2$ $out[2][2] := ((kq[2][2])^{-2} - 1) / 2$ $out[3][3] := ((kq[3][3])^{-2} - 1) / 2$ others ud	$out[1][1] := ((1 + kq[1][1])^2 - 1) / 2$ $out[2][2] := ((1 + kq[2][2])^2 - 1) / 2$ $out[3][3] := ((1 + kq[3][3])^2 - 1) / 2$ others ud	$out[1][1] := ((kq[1][1] / (kq[1][1] - 1))^2 - 1) / 2$ $out[2][2] := ((kq[2][2] / (kq[2][2] - 1))^2 - 1) / 2$ $out[3][3] := ((kq[3][3] / (kq[3][3] - 1))^2 - 1) / 2$ others ud

G Table D-24 out for lfst_g_knownquantity

$H_1 \wedge H_2 \rightarrow G$

Continue →

H_1

	cdt	gdt	lfst	efst	list
	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$	kq	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$
$kq[1][1]=\text{ud} \wedge kq[2][2]=\text{ud} \wedge kq[3][3]=\text{ud}$					
$kq[1][1]=\text{ud} \wedge kq[2][2]=\text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[3][3] := (kq[3][3]^{-1} - 1) / 2$ others ud	$\text{out}[3][3] := (kq[3][3] - 1) / 2$ others ud	kq	$\text{out}[3][3] := kq[3][3] / (1 - 2 * kq[3][3])$ others ud	$\text{out}[3][3] := (kq[3][3]^2 + 2 * kq[2][2]) / 2$ others ud
$kq[1][1]=\text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3]=\text{ud}$	$\text{out}[2][2] := (kq[2][2]^{-1} - 1) / 2$ others ud	$\text{out}[2][2] := (kq[2][2] - 1) / 2$ others ud	kq	$\text{out}[2][2] := kq[2][2] / (1 - 2 * kq[2][2])$ others ud	$\text{out}[2][2] := (kq[2][2]^2 + 2 * kq[2][2]) / 2$ others ud
$kq[1][1]=\text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[2][2] := (kq[2][2]^{-1} - 1) / 2$ $\text{out}[3][3] := (kq[3][3]^{-1} - 1) / 2$ others ud	$\text{out}[2][2] := (kq[2][2] - 1) / 2$ $\text{out}[3][3] := (kq[3][3] - 1) / 2$ others ud	kq	$\text{out}[2][2] := kq[2][2] / (1 - 2 * kq[2][2])$ $\text{out}[3][3] := kq[3][3] / (1 - 2 * kq[3][3])$ others ud	$\text{out}[2][2] := (kq[2][2]^2 + 2 * kq[2][2]) / 2$ $\text{out}[3][3] := (kq[3][3]^2 + 2 * kq[2][2]) / 2$ others ud
$kq[1][1] \neq \text{ud} \wedge kq[2][2]=\text{ud} \wedge kq[3][3]=\text{ud}$	$\text{out}[1][1] := (kq[1][1]^{-1} - 1) / 2$ others ud	$\text{out}[1][1] := (kq[1][1] - 1) / 2$ others ud	kq	$\text{out}[1][1] := kq[1][1] / (1 - 2 * kq[1][1])$ others ud	$\text{out}[1][1] := (kq[1][1]^2 + 2 * kq[1][1]) / 2$ others ud
$kq[1][1] \neq \text{ud} \wedge kq[2][2]=\text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[1][1] := (kq[1][1]^{-1} - 1) / 2$ $\text{out}[3][3] := (kq[3][3]^{-1} - 1) / 2$ others ud	$\text{out}[1][1] := (kq[1][1] - 1) / 2$ $\text{out}[3][3] := (kq[3][3] - 1) / 2$ others ud	kq	$\text{out}[1][1] := kq[1][1] / (1 - 2 * kq[1][1])$ $\text{out}[3][3] := kq[3][3] / (1 - 2 * kq[3][3])$ others ud	$\text{out}[1][1] := (kq[1][1]^2 + 2 * kq[1][1]) / 2$ $\text{out}[3][3] := (kq[3][3]^2 + 2 * kq[2][2]) / 2$ others ud
$kq[1][1] \neq \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3]=\text{ud}$	$\text{out}[1][1] := (kq[1][1]^{-1} - 1) / 2$ $\text{out}[2][2] := (kq[2][2]^{-1} - 1) / 2$ others ud	$\text{out}[1][1] := (kq[1][1] - 1) / 2$ $\text{out}[2][2] := (kq[2][2] - 1) / 2$ others ud	kq	$\text{out}[1][1] := kq[1][1] / (1 - 2 * kq[1][1])$ $\text{out}[2][2] := kq[2][2] / (1 - 2 * kq[2][2])$ others ud	$\text{out}[1][1] := (kq[1][1]^2 + 2 * kq[1][1]) / 2$ $\text{out}[2][2] := (kq[2][2]^2 + 2 * kq[2][2]) / 2$ others ud
$kq[1][1] \neq \text{ud} \wedge kq[2][2] \neq \text{ud} \wedge kq[3][3] \neq \text{ud}$	$\text{out}[1][1] := (kq[1][1]^{-1} - 1) / 2$ $\text{out}[2][2] := (kq[2][2]^{-1} - 1) / 2$ $\text{out}[3][3] := (kq[3][3]^{-1} - 1) / 2$ others ud	$\text{out}[1][1] := (kq[1][1] - 1) / 2$ $\text{out}[2][2] := (kq[2][2] - 1) / 2$ $\text{out}[3][3] := (kq[3][3] - 1) / 2$ others ud	kq	$\text{out}[1][1] := kq[1][1] / (1 - 2 * kq[1][1])$ $\text{out}[2][2] := kq[2][2] / (1 - 2 * kq[2][2])$ $\text{out}[3][3] := kq[3][3] / (1 - 2 * kq[3][3])$ others ud	$\text{out}[1][1] := (kq[1][1]^2 + 2 * kq[1][1]) / 2$ $\text{out}[2][2] := (kq[2][2]^2 + 2 * kq[2][2]) / 2$ $\text{out}[3][3] := (kq[3][3]^2 + 2 * kq[2][2]) / 2$ others ud

→Continue

G

Table D-24 out for lfst_g_knownquantity (Continue)

$H_1 \wedge H_2 \rightarrow G$

→Continue

H₂ : kgflag

H₁

	eist	tst
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]=ud	Out:=[ud]	Out:=[ud]
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[3][3] := ((kq[3][3]/(kq[3][3]-1))^2 - 1) / 2 others ud.	out[3][3] := (exp(2 * kq[3][3]) - 1) / 2 others ud
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[2][2] := ((kq[2][2]/(kq[2][2]-1))^2 - 1) / 2 others ud.	out[2][2] := (exp(2 * kq[2][2]) - 1) / 2 others ud
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[2][2] := ((kq[2][2]/(kq[2][2]-1))^2 - 1) / 2 out[3][3] := ((kq[3][3]/(kq[3][3]-1))^2 - 1) / 2 others ud	out[2][2] := (exp(2 * kq[2][2]) - 1) / 2 out[3][3] := (exp(2 * kq[3][3]) - 1) / 2 others ud
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]=ud	out[1][1] := ((kq[1][1]/(kq[1][1]-1))^2 - 1) / 2 others ud.	out[1][1] := (exp(2 * kq[1][1]) - 1) / 2 others ud
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[1][1] := ((kq[1][1]/(kq[1][1]-1))^2 - 1) / 2 out[3][3] := ((kq[3][3]/(kq[3][3]-1))^2 - 1) / 2 others ud	out[1][1] := (exp(2 * kq[1][1]) - 1) / 2 out[3][3] := (exp(2 * kq[3][3]) - 1) / 2 others ud
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[1][1] := ((kq[1][1]/(kq[1][1]-1))^2 - 1) / 2 out[2][2] := ((kq[2][2]/(kq[2][2]-1))^2 - 1) / 2 others ud	out[1][1] := (exp(2 * kq[1][1]) - 1) / 2 out[2][2] := (exp(2 * kq[2][2]) - 1) / 2 others ud
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[1][1] := ((kq[1][1]/(kq[1][1]-1))^2 - 1) / 2 out[2][2] := ((kq[2][2]/(kq[2][2]-1))^2 - 1) / 2 out[3][3] := ((kq[3][3]/(kq[3][3]-1))^2 - 1) / 2 others ud	out[1][1] := (exp(2 * kq[1][1]) - 1) / 2 out[2][2] := (exp(2 * kq[2][2]) - 1) / 2 out[3][3] := (exp(2 * kq[3][3]) - 1) / 2 others ud

Continue→

G Table D-24 out for lfst_g_knownquantity(Continue)

H₁ ∧ H₂ → G

H	disp.dispu=ud^ disp.dispv=ud^ disp.dispw=ud	Out :=[ud]
	disp.dispu=ud^ disp.dispv=ud^ disp.dispw≠ud	out[3][3] := ((1 + disp.dispw / sg.width) ² - 1) / 2 others ud
	disp.dispu=ud^ disp.dispv≠ud^ disp.dispw=ud	out[2][2] := ((1 + disp.dispv / sg.height) ² - 1) / 2 others ud
	disp.dispu=ud^ disp.dispv≠ud^ disp.dispw≠ud	out[2][2] := ((1 + disp.dispv / sg.height) ² - 1) / 2 out[3][3] := ((1 + disp.dispw / sg.width) ² - 1) / 2 others ud
	disp.dispu≠ud^ disp.dispv=ud^ disp.dispw=ud	out[1][1] := ((1 + disp.dispu / sg.length) ² - 1) / 2 others ud
	disp.dispu≠ud^ disp.dispv=ud^ disp.dispw≠ud	out[1][1] := ((1 + disp.dispu / sg.length) ² - 1) / 2 out[3][3] := ((1 + disp.dispw / sg.width) ² - 1) / 2 others ud
	disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw=ud	out[1][1] := ((1 + disp.dispu / sg.length) ² - 1) / 2 out[2][2] := ((1 + disp.dispv / sg.height) ² - 1) / 2 others ud
	disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw≠ud	out[1][1] := ((1 + disp.dispu / sg.length) ² - 1) / 2 out[2][2] := ((1 + disp.dispv / sg.height) ² - 1) / 2 out[3][3] := ((1 + disp.dispw / sg.width) ² - 1) / 2 others ud

Table D-25 out for lfst_g_geometry

D. 3.22 Eulerian (Almansi's) finite strain tensor module

Prefix: `efst_`

Reference: MG – C. 3.2.19

D. 3.22.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

TensorFlagT from the tensor data definition module

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
<code>efst_g_knownquantity</code>	TensorDataT TensorFlagT	TensorDataT	
<code>efst_g_geometry</code>	SpecimenGeometryT DisplacementT	TensorDataT	none

D. 3.22.2 Interface semantics

State variable: none

State invariant: none

Assumption:

Access routine semantics

`lfst_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)`

Exception: Table D-26

Output: Table D-27

`lfst_g_geometry(sg: SpecimenGeometryT, disp: DisplacementT)`

Exception: none

Output: Table D-28

H_1	mdg	sdg	mdpg	sdpg	cdt	gdt	lfst	efst	list	eist	tst
G	(kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0) ⇒ d_zero non	non	(kq[1][1]=-1)∨ (kq[2][2]=-1)∨ (kq[3][3]=-1) ⇒ d_zero	non	non	(kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0) ⇒ d_zero	(kq[1][1]=-0.5)∨ (kq[2][2]=-0.5)∨ (kq[3][3]=-0.5) ⇒ d_zero	non	(kq[1][1]=-1)∨ (kq[2][2]=-1)∨ (kq[3][3]=-1) ⇒ d_zero	non	non

$H_1 \rightarrow G$

Table D-26 exception for efst_g_knownquantity

$H_2 : \text{kgflag}$

H_1

	mdg	sdg	mdpg	sdpg
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := (1 - kq[3][3]^{-2}) / 2$ others ud	$out[3][3] := (1 - kq[3][3]^2) / 2$ others ud	$out[3][3] := (1 - (1 + kq[3][3])^{-2}) / 2$ others ud.	$out[3][3] := (1 - (1 - kq[3][3])^2) / 2$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[2][2] := (1 - kq[2][2]^{-2}) / 2$ others ud	$out[2][2] := (1 - kq[2][2]^2) / 2$ others ud	$out[2][2] := (1 - (1 + kq[2][2])^{-2}) / 2$ others ud.	$out[2][2] := (1 - (1 - kq[2][2])^2) / 2$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := (1 - kq[2][2]^{-2}) / 2$ $out[3][3] := (1 - kq[3][3]^{-2}) / 2$ others ud	$out[2][2] := (1 - kq[2][2]^2) / 2$ $out[3][3] := (1 - kq[3][3]^2) / 2$ others ud	$out[2][2] := (1 - (1 + kq[2][2])^{-2}) / 2$ $out[3][3] := (1 - (1 + kq[3][3])^{-2}) / 2$ others ud	$out[2][2] := (1 - (1 - kq[2][2])^2) / 2$ $out[3][3] := (1 - (1 - kq[3][3])^2) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := (1 - kq[1][1]^{-2}) / 2$ others ud	$out[1][1] := (1 - kq[1][1]^2) / 2$ others ud	$out[1][1] := (1 - (1 + kq[1][1])^{-2}) / 2$ others ud.	$out[1][1] := (1 - (1 - kq[1][1])^2) / 2$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := (1 - kq[1][1]^{-2}) / 2$ $out[3][3] := (1 - kq[3][3]^{-2}) / 2$ others ud	$out[1][1] := (1 - kq[1][1]^2) / 2$ $out[3][3] := (1 - kq[3][3]^2) / 2$ others ud	$out[1][1] := (1 - (1 + kq[1][1])^{-2}) / 2$ $out[3][3] := (1 - (1 + kq[3][3])^{-2}) / 2$ others ud	$out[1][1] := (1 - (1 - kq[1][1])^2) / 2$ $out[3][3] := (1 - (1 - kq[3][3])^2) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := (1 - kq[1][1]^{-2}) / 2$ $out[2][2] := (1 - kq[2][2]^{-2}) / 2$ others ud	$out[1][1] := (1 - kq[1][1]^2) / 2$ $out[2][2] := (1 - kq[2][2]^2) / 2$ others ud	$out[1][1] := (1 - (1 + kq[1][1])^{-2}) / 2$ $out[2][2] := (1 - (1 + kq[2][2])^{-2}) / 2$ others ud	$out[1][1] := (1 - (1 - kq[1][1])^2) / 2$ $out[2][2] := (1 - (1 - kq[2][2])^2) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := (1 - kq[1][1]^{-2}) / 2$ $out[2][2] := (1 - kq[2][2]^{-2}) / 2$ $out[3][3] := (1 - kq[3][3]^{-2}) / 2$ others ud	$out[1][1] := (1 - kq[1][1]^2) / 2$ $out[2][2] := (1 - kq[2][2]^2) / 2$ $out[3][3] := (1 - kq[3][3]^2) / 2$ others ud	$out[1][1] := (1 - (1 + kq[1][1])^{-2}) / 2$ $out[2][2] := (1 - (1 + kq[2][2])^{-2}) / 2$ $out[3][3] := (1 - (1 + kq[3][3])^{-2}) / 2$ others ud	$out[1][1] := (1 - (1 - kq[1][1])^2) / 2$ $out[2][2] := (1 - (1 - kq[2][2])^2) / 2$ $out[3][3] := (1 - (1 - kq[3][3])^2) / 2$ others ud

G Table D-27 out for $\text{efst_g_knownquantity}$

$H_1 \wedge H_2 \rightarrow G$

Continue→

$H_2 : \text{kgflag}$

H_1

	cdt	gdt	lfst	efst	list
	Out:= [ud]	Out:=[ud]	Out:=[ud]	kq	Out:=[ud]
	$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$				
	$out[3][3] := (1 - kq[3][3])/2$ others ud	$out[3][3] := (1 - kq[3][3]^{-1})/2$ others ud	$out[3][3] := kq[3][3]/(1 + 2 * kq[3][3])$ others ud	kq	$out[3][3] := (1 - (1 + kq[3][3])^{-2})/2$ others ud.
	$out[2][2] := (1 - kq[2][2])/2$ others ud	$out[2][2] := (1 - kq[2][2]^{-1})/2$ others ud	$out[2][2] := kq[2][2]/(1 + 2 * kq[2][2])$ others ud	kq	$out[2][2] := (1 - (1 + kq[2][2])^{-2})/2$ others ud.
	$out[2][2] := (1 - kq[2][2])/2$ $out[3][3] := (1 - kq[3][3])/2$ others ud	$out[2][2] := (1 - kq[2][2]^{-1})/2$ $out[3][3] := (1 - kq[3][3]^{-1})/2$ others ud	$out[2][2] := kq[2][2]/(1 + 2 * kq[2][2])$ $out[3][3] := kq[3][3]/(1 + 2 * kq[3][3])$ others ud	kq	$out[2][2] := (1 - (1 + kq[2][2])^{-2})/2$ $out[3][3] := (1 - (1 + kq[3][3])^{-2})/2$ others ud
	$out[1][1] := (1 - kq[1][1])/2$ others ud	$out[1][1] := (1 - kq[1][1]^{-1})/2$ others ud	$out[1][1] := kq[1][1]/(1 + 2 * kq[1][1])$ others ud	kq	$out[1][1] := (1 - (1 + kq[1][1])^{-2})/2$ others ud.
	$out[1][1] := (1 - kq[1][1])/2$ $out[3][3] := (1 - kq[3][3])/2$ others ud	$out[1][1] := (1 - kq[1][1]^{-1})/2$ $out[3][3] := (1 - kq[3][3]^{-1})/2$ others ud	$out[1][1] := kq[1][1]/(1 + 2 * kq[1][1])$ $out[3][3] := kq[3][3]/(1 + 2 * kq[3][3])$ others ud	kq	$out[1][1] := (1 - (1 + kq[1][1])^{-2})/2$ $out[3][3] := (1 - (1 + kq[3][3])^{-2})/2$ others ud
	$out[1][1] := (1 - kq[1][1])/2$ $out[2][2] := (1 - kq[2][2])/2$ others ud	$out[1][1] := (1 - kq[1][1]^{-1})/2$ $out[2][2] := (1 - kq[2][2]^{-1})/2$ others ud	$out[1][1] := kq[1][1]/(1 + 2 * kq[1][1])$ $out[2][2] := kq[2][2]/(1 + 2 * kq[2][2])$ others ud	kq	$out[1][1] := (1 - (1 + kq[1][1])^{-2})/2$ $out[2][2] := (1 - (1 + kq[2][2])^{-2})/2$ others ud
	$out[1][1] := (1 - kq[1][1])/2$ $out[2][2] := (1 - kq[2][2])/2$ $out[3][3] := (1 - kq[3][3])/2$ others ud	$out[1][1] := (1 - kq[1][1]^{-1})/2$ $out[2][2] := (1 - kq[2][2]^{-1})/2$ $out[3][3] := (1 - kq[3][3]^{-1})/2$ others ud	$out[1][1] := kq[1][1]/(1 + 2 * kq[1][1])$ $out[2][2] := kq[2][2]/(1 + 2 * kq[2][2])$ $out[3][3] := kq[3][3]/(1 + 2 * kq[3][3])$ others ud	kq	$out[1][1] := (1 - (1 + kq[1][1])^{-2})/2$ $out[2][2] := (1 - (1 + kq[2][2])^{-2})/2$ $out[3][3] := (1 - (1 + kq[3][3])^{-2})/2$ othrs ud

→Continue

G

Table D-27 out for efst_knownquantity (Continue)

$H_1 \wedge H_2 \rightarrow G$

Continue→

H_2 : kgflag

H_1

	eist	tst
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	Out:=[ud]	Out:=[ud]
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := (1 - (1 - kq[3][3])^2) / 2$ others ud.	$out[3][3] := (1 - \exp(-2 * kq[3][3])) / 2$ others ud
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[2][2] := (1 - (1 - kq[2][2])^2) / 2$ others ud.	$out[2][2] := (1 - \exp(-2 * kq[2][2])) / 2$ others ud
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := (1 - (1 - kq[2][2])^2) / 2$ $out[3][3] := (1 - (1 - kq[3][3])^2) / 2$ others ud	$out[2][2] := (1 - \exp(-2 * kq[2][2])) / 2$ $out[3][3] := (1 - \exp(-2 * kq[3][3])) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := (1 - (1 - kq[1][1])^2) / 2$ others ud.	$out[1][1] := (1 - \exp(-2 * kq[1][1])) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := (1 - (1 - kq[1][1])^2) / 2$ $out[3][3] := (1 - (1 - kq[3][3])^2) / 2$ others ud	$out[1][1] := (1 - \exp(-2 * kq[1][1])) / 2$ $out[3][3] := (1 - \exp(-2 * kq[3][3])) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := (1 - (1 - kq[1][1])^2) / 2$ $out[2][2] := (1 - (1 - kq[2][2])^2) / 2$ others ud	$out[1][1] := (1 - \exp(-2 * kq[1][1])) / 2$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := (1 - (1 - kq[1][1])^2) / 2$ $out[2][2] := (1 - (1 - kq[2][2])^2) / 2$ $out[3][3] := (1 - (1 - kq[3][3])^2) / 2$ others ud	$out[1][1] := (1 - \exp(-2 * kq[1][1])) / 2$ $out[2][2] := (1 - \exp(-2 * kq[2][2])) / 2$ $out[3][3] := (1 - \exp(-2 * kq[3][3])) / 2$ others ud

Continue →

G Table D-27 out for efst_g_knownquantity

$H_1 \wedge H_2 \rightarrow G$

G

H→G

H	$\text{disp.dispu}=\text{ud}\wedge$ $\text{disp.dispv}=\text{ud}\wedge$ $\text{disp.dispw}=\text{ud}$	Out :=[ud]
	$\text{disp.dispu}=\text{ud}\wedge$ $\text{disp.dispv}=\text{ud}\wedge$ $\text{disp.dispw}\neq\text{ud}$	$\text{out}[3][3] := (1 - (1 + \text{disp.dispw} / \text{sg.width})^{-2}) / 2$ others ud
	$\text{disp.dispu}=\text{ud}\wedge$ $\text{disp.dispv}\neq\text{ud}\wedge$ $\text{disp.dispw}=\text{ud}$	$\text{out}[2][2] := (1 - (1 + \text{disp.dispv} / \text{sg.height})^{-2}) / 2$ others ud
	$\text{disp.dispu}=\text{ud}\wedge$ $\text{disp.dispv}\neq\text{ud}\wedge$ $\text{disp.dispw}\neq\text{ud}$	$\text{out}[2][2] := (1 - (1 + \text{disp.dispv} / \text{sg.height})^{-2}) / 2$ $\text{out}[3][3] := (1 - (1 + \text{disp.dispw} / \text{sg.width})^{-2}) / 2$ others ud
	$\text{disp.dispu}\neq\text{ud}\wedge$ $\text{disp.dispv}=\text{ud}\wedge$ $\text{disp.dispw}=\text{ud}$	$\text{out}[1][1] := (1 - (1 + \text{disp.dispv} / \text{sg.height})^{-2}) / 2$ others ud
	$\text{disp.dispu}\neq\text{ud}\wedge$ $\text{disp.dispv}=\text{ud}\wedge$ $\text{disp.dispw}\neq\text{ud}$	$\text{out}[1][1] := (1 - (1 + \text{disp.dispv} / \text{sg.height})^{-2}) / 2$ others ud
	$\text{disp.dispu}\neq\text{ud}\wedge$ $\text{disp.dispv}\neq\text{ud}\wedge$ $\text{disp.dispw}=\text{ud}$	$\text{out}[1][1] := (1 - (1 + \text{disp.dispv} / \text{sg.height})^{-2}) / 2$ $\text{out}[2][2] := (1 - (1 + \text{disp.dispv} / \text{sg.height})^{-2}) / 2$ others ud
	$\text{disp.dispu}\neq\text{ud}\wedge$ $\text{disp.dispv}\neq\text{ud}\wedge$ $\text{disp.dispw}\neq\text{ud}$	$\text{out}[1][1] := (1 - (1 + \text{disp.dispv} / \text{sg.height})^{-2}) / 2$ $\text{out}[2][2] := (1 - (1 + \text{disp.dispv} / \text{sg.height})^{-2}) / 2$ $\text{out}[3][3] := (1 - (1 + \text{disp.dispw} / \text{sg.width})^{-2}) / 2$ others ud

Table D-28 out for `efst_g_geometry`

D. 3.23 Lagrangian (Green's) infinitesimal strain tensor module

Prefix: list_

Reference: MG – C. 3.2.18

D. 3.23.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

TensorFlagT from the tensor data definition module

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
list_g_knownquantity	TensorDataT TensorFlagT	TensorDataT	
list_g_geometry	SpecimenGeometryT DisplacementT	TensorDataT	none

D. 3.23.2 Interface semantics

State variable: none

State invariant: none

Assumption:

Access routine semantics

list_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)

Exception: Table D-29

Output: Table D-30

list_g_geometry(sg: SpecimenGeometryT, disp: DisplacementT)

Exception: none

Output: Table D-31

H ₁	mdg	sdg	mdpg	sdpg	cdt	gdt	lfst	efst	list	eist	tst
G	non	(kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0) ⇒ d_zero	non	(kq[1][1]=1)∨ (kq[2][2]=1)∨ (kq[3][3]=1) ⇒ d_zero	(kq[1][1]<0)∨ (kq[2][2]<0)∨ (kq[3][3]<0) ⇒ sr_lesszero (kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0) ⇒ d_zero	(kq[1][1]<0)∨ (kq[2][2]<0)∨ (kq[3][3]<0) ⇒ sr_lesszero,	(kq[1][1]<0.5)∨ (kq[2][2]<0.5)∨ (kq[3][3]<0.5) ⇒ sr_lesszero,	(kq[1][1]>0.5)∨ (kq[2][2]>0.5)∨ (kq[3][3]>0.5) ⇒ sr_lesszero, (kq[1][1]=0.5)∨ (kq[2][2]=0.5)∨ (kq[3][3]=0.5) ⇒ d_zero	non	(kq[1][1]=1) ∨ (kq[2][2]=1) ∨ (kq[3][3]=1) ⇒ d_zero	non

H₁ →G

Table D-29 exception for list_g_knownquantity

H_2 : kgflag

H_1

	mdg	sdg	mdpg	sdpg	cdt	gdt
	Out:=[ud]	Out:=[ud]	kq	Out:=[ud]	Out:=[ud]	Out:=[ud]
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$						
$kq[1][1]=ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[3][3] := kq[3][3] - 1$ others ud	$out[3][3] := 1 / kq[3][3] - 1$ others ud	kq	$out[3][3] := kq[3][3] / (1 - kq[3][3])$ others ud.	$out[3][3] := 1 / \sqrt{kq[3][3]} - 1$ others ud	$out[3][3] := \sqrt{kq[3][3]} - 1$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$kq[2][2] := kq[2][2] - 1$ others ud	$out[2][2] := 1 / kq[2][2] - 1$ others ud	kq	$out[2][2] := kq[2][2] / (1 - kq[2][2])$ others ud.	$out[2][2] := 1 / \sqrt{kq[2][2]} - 1$ others ud	$out[2][2] := \sqrt{kq[2][2]} - 1$ others ud.
$kq[1][1]=ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[2][2] := kq[2][2] - 1$ $out[3][3] := kq[3][3] - 1$ others ud	$out[2][2] := 1 / kq[2][2] - 1$ $out[3][3] := 1 / kq[3][3] - 1$ others ud	kq	$out[2][2] := kq[2][2] / (1 - kq[2][2])$ $out[3][3] := kq[3][3] / (1 - kq[3][3])$ others ud.	$out[2][2] := 1 / \sqrt{kq[2][2]} - 1$ $out[3][3] := 1 / \sqrt{kq[3][3]} - 1$ others ud	$out[2][2] := \sqrt{kq[2][2]} - 1$ $out[3][3] := \sqrt{kq[3][3]} - 1$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3]=ud$	$out[1][1] := kq[1][1] - 1$ others ud	$out[1][1] := 1 / kq[1][1] - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]} - 1$ others ud	$out[1][1] := \sqrt{kq[1][1]} - 1$ others ud.
$kq[1][1] \neq ud \wedge$ $kq[2][2]=ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := kq[1][1] - 1$ $out[3][3] := kq[3][3] - 1$ others ud	$out[1][1] := 1 / kq[1][1] - 1$ $out[3][3] := 1 / kq[3][3] - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ $out[3][3] := kq[3][3] / (1 - kq[3][3])$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]} - 1$ $out[3][3] := 1 / \sqrt{kq[3][3]} - 1$ others ud	$out[1][1] := \sqrt{kq[1][1]} - 1$ $out[3][3] := \sqrt{kq[3][3]} - 1$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3]=ud$	$out[1][1] := kq[1][1] - 1$ $out[2][2] := kq[2][2] - 1$ others ud	$out[1][1] := 1 / kq[1][1] - 1$ $out[2][2] := 1 / kq[2][2] - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ $out[2][2] := kq[2][2] / (1 - kq[2][2])$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]} - 1$ $out[2][2] := 1 / \sqrt{kq[2][2]} - 1$ others ud	$out[1][1] := \sqrt{kq[1][1]} - 1$ $out[2][2] := \sqrt{kq[2][2]} - 1$ others ud
$kq[1][1] \neq ud \wedge$ $kq[2][2] \neq ud \wedge$ $kq[3][3] \neq ud$	$out[1][1] := kq[1][1] - 1$ $out[2][2] := kq[2][2] - 1$ $out[3][3] := kq[3][3] - 1$ others ud	$out[1][1] := 1 / kq[1][1] - 1$ $out[2][2] := 1 / kq[2][2] - 1$ $out[3][3] := 1 / kq[3][3] - 1$ others ud	kq	$out[1][1] := kq[1][1] / (1 - kq[1][1])$ $out[2][2] := kq[2][2] / (1 - kq[2][2])$ $out[3][3] := kq[3][3] / (1 - kq[3][3])$ others ud.	$out[1][1] := 1 / \sqrt{kq[1][1]} - 1$ $out[2][2] := 1 / \sqrt{kq[2][2]} - 1$ $out[3][3] := 1 / \sqrt{kq[3][3]} - 1$ others ud	$out[1][1] := \sqrt{kq[1][1]} - 1$ $out[2][2] := \sqrt{kq[2][2]} - 1$ $out[3][3] := \sqrt{kq[3][3]} - 1$ others ud

G

Table D-30 out for list_g_knownquantity

$H_1 \wedge H_2 \rightarrow G$

Continue→

H₁

	lfst	efst	list	eist	tst	
	kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]=ud	Out:=[ud]		kq	Out:=[ud]	Out:=[ud]
	kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[3][3] := sqrt(2 * kq[3][3] + 1) - 1 others ud.		kq	out[3][3] := kq[3][3]/(1 - kq[3]) others ud.	out[3][3] := exp(kq[3][3]) - 1 others ud.
	kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[2][2] := sqrt(2 * kq[2][2] + 1) - 1 others ud.		kq	out[2][2] := kq[2][2]/(1 - kq[2]) others ud.	out[2][2] := exp(kq[2][2]) - 1 others ud.
	kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[2][2] := sqrt(2 * kq[2][2] + 1) - 1 out[3][3] := sqrt(2 * kq[3][3] + 1) - 1 others ud.		kq	out[2][2] := kq[2][2]/(1 - kq[2]) out[3][3] := kq[3][3]/(1 - kq[3]) others ud.	out[2][2] := exp(kq[2][2]) - 1 out[3][3] := exp(kq[3][3]) - 1 others ud.
	kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]=ud	out[1][1] := sqrt(2 * kq[1][1] + 1) - 1 others ud.		kq	out[1][1] := kq[1][1]/(1 - kq[1][1]) others ud.	out[1][1] := exp(kq[1][1]) - 1 others ud.
	kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[1][1] := sqrt(2 * kq[1][1] + 1) - 1 out[3][3] := sqrt(2 * kq[3][3] + 1) - 1 others ud.		kq	out[1][1] := kq[1][1]/(1 - kq[1][1]) out[3][3] := kq[3][3]/(1 - kq[3]) others ud.	out[1][1] := exp(kq[1][1]) - 1 out[3][3] := exp(kq[3][3]) - 1 others ud.
	kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[1][1] := sqrt(2 * kq[1][1] + 1) - 1 out[2][2] := sqrt(2 * kq[2][2] + 1) - 1 others ud.		kq	out[1][1] := kq[1][1]/(1 - kq[1][1]) out[2][2] := kq[2][2]/(1 - kq[2]) others ud.	out[1][1] := exp(kq[1][1]) - 1 out[2][2] := exp(kq[2][2]) - 1 others ud.
	kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[1][1] := sqrt(2 * kq[1][1] + 1) - 1 out[2][2] := sqrt(2 * kq[2][2] + 1) - 1 out[3][3] := sqrt(2 * kq[3][3] + 1) - 1 others ud.		kq	out[1][1] := kq[1][1]/(1 - kq[1][1]) out[2][2] := kq[2][2]/(1 - kq[2]) out[3][3] := kq[3][3]/(1 - kq[3]) others ud.	out[1][1] := exp(kq[1][1]) - 1 out[2][2] := exp(kq[2][2]) - 1 out[3][3] := exp(kq[3][3]) - 1 others ud.

→Continue

G

Table D-30 out for list_g_knownquantity (Continue)

H₁ ∧ H₂ → G

G

H→G

H

disp.dispu=ud^ disp.dispv=ud^ disp.dispw=ud	Out :=[ud]
disp.dispu=ud^ disp.dispv=ud^ disp.dispw≠ud	out[3][3] := disp.dispw / sg.width others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw=ud	out[2][2] := disp.dispv / sg.height others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw≠ud	out[2][2] := disp.dispv / sg.height out[3][3] := disp.dispw / sg.width others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw=ud	out[1][1] := disp.dispu / sg.length others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw≠ud	out[1][1] := disp.dispu / sg.length out[3][3] := disp.dispw / sg.width others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw=ud	out[1][1] := disp.dispu / sg.length out[2][2] := disp.dispv / sg.height others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw≠ud	out[1][1] := disp.dispu / sg.length out[2][2] := disp.dispv / sg.height out[3][3] := disp.dispw / sg.width others ud

Table D-31 out for list_g_geometry

D. 3.24 Eulerian (Almansi's) infinitesimal strain tensor module

Prefix: eist_

Reference: MG – C. 3.2.17

D. 3.24.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

TensorFlagT from the tensor data definition module

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
eist_g_knownquantity	TensorDataT TensorFlagT	TensorDataT	
eist_g_geometry	SpecimenGeometryT DisplacementT	TensorDataT	none

D. 3.24.2 Interface semantics

State variable: none

State invariant: none

Assumption:

Access routine semantics

eist_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)

exception: Table D-32

output: Table D-33

eist_g_geometry(sg: SpecimenGeometryT, disp: DisplacementT)

Exception: none

Output: Table D-34

	mdg	sdg	mdpg	sdpg	cdt	gdt	lfst	efst	list	eist	tst
H ₁	(kq[1][1]=0)∨ (kq[2][2]=0)∨ (kq[3][3]=0) ⇒ d_zero	non	(kq[1][1]=1)∨ (kq[2][2]=1)∨ (kq[3][3]=1) ⇒ d_zero	non	(kq[1][1]<0) ∨ (kq[2][2]<0) ∨ (kq[3][3]<0) ⇒ sr_lesszero	(kq[1][1]<0) ∨ (kq[2][2]<0) ∨ (kq[3][3]<0) ⇒ sr_lesszero, (kq[1][1]=0) ∨ (kq[2][2]=0) ∨ (kq[3][3]=0) ⇒ d_zero	(kq[1][1]<-0.5) ∨ (kq[2][2]<-0.5) ∨ (kq[3][3]<-0.5) ⇒ sr_lesszero, (kq[1][1]=-0.5) ∨ (kq[2][2]=-0.5) ∨ (kq[3][3]=-0.5) ⇒ d_zero	(kq[1][1]>0.5) ∨ (kq[2][2]>0.5) ∨ (kq[3][3]>0.5) ⇒ sr_lesszero,	(kq[1][1]=-1) ∨ (kq[2][2]=-1) ∨ (kq[3][3]=-1) ⇒ d_zero non	non	non
G											

Table D-32 exception for eist_g_knownquantity

H₁ → G

H₁

	mdg	sdg	mdpg	sdpg	cdt	gdt
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]=ud	Out:=[ud]	Out:=[ud]	Out:=[ud]	kq	Out:=[ud]	Out:=[ud]
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[3][3] := 1 - 1 / kq[3][3] others ud	out[3][3] := 1 - kq[3][3] others ud	out[3][3] := kq[3][3] / (1 + kq[3][3]) others ud.	kq	out[3][3] := 1 - sqrt(kq[3][3]) others ud	out[3][3] := 1 - 1 / sqrt(kq[3][3]) others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]=ud	kq[2][2] := 1 - 1 / kq[2][2] others ud	out[2][2] := 1 - kq[2][2] others ud	out[2][2] := kq[2][2] / (1 + kq[2][2]) others ud.	kq	out[2][2] := 1 - sqrt(kq[2][2]) others ud	out[2][2] := 1 - 1 / sqrt(kq[2][2]) others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[2][2] := 1 - 1 / kq[2][2] out[3][3] := 1 - 1 / kq[3][3] others ud	out[2][2] := 1 - kq[2][2] out[3][3] := 1 - kq[3][3] others ud	out[2][2] := kq[2][2] / (1 + kq[2][2]) out[3][3] := kq[3][3] / (1 + kq[3][3]) others ud.	kq	out[2][2] := 1 - sqrt(kq[2][2]) out[3][3] := 1 - sqrt(kq[3][3]) others ud	out[2][2] := 1 - 1 / sqrt(kq[2][2]) out[3][3] := 1 - 1 / sqrt(kq[3][3]) others ud.
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]=ud	out[1][1] := 1 - 1 / kq[1][1] others ud	out[1][1] := 1 - kq[1][1] others ud	out[1][1] := kq[1][1] / (1 + kq[1][1]) others ud.	kq	out[1][1] := 1 - sqrt(kq[1][1]) others ud	out[1][1] := 1 - 1 / sqrt(kq[1][1]) others ud.
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[1][1] := 1 - 1 / kq[1][1] out[3][3] := 1 - 1 / kq[3][3] others ud	out[1][1] := 1 - kq[1][1] out[3][3] := 1 - kq[3][3] others ud	out[1][1] := kq[1][1] / (1 + kq[1][1]) out[3][3] := kq[3][3] / (1 + kq[3][3]) others ud.	kq	out[1][1] := 1 - sqrt(kq[1][1]) out[3][3] := 1 - sqrt(kq[3][3]) others ud	out[1][1] := 1 - 1 / sqrt(kq[1][1]) out[3][3] := 1 - 1 / sqrt(kq[3][3]) others ud.
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[1][1] := 1 - 1 / kq[1][1] out[2][2] := 1 - 1 / kq[2][2] others ud	out[1][1] := 1 - kq[1][1] out[2][2] := 1 - kq[2][2] others ud	out[1][1] := kq[1][1] / (1 + kq[1][1]) out[2][2] := kq[2][2] / (1 + kq[2][2]) others ud.	kq	out[1][1] := 1 - sqrt(kq[1][1]) out[2][2] := 1 - sqrt(kq[2][2]) others ud	out[1][1] := 1 - 1 / sqrt(kq[1][1]) out[2][2] := 1 - 1 / sqrt(kq[2][2]) others ud.
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[1][1] := 1 - 1 / kq[1][1] out[2][2] := 1 - 1 / kq[2][2] out[3][3] := 1 - 1 / kq[3][3] others ud	out[1][1] := 1 - kq[1][1] out[2][2] := 1 - kq[2][2] out[3][3] := 1 - kq[3][3] others ud	out[1][1] := kq[1][1] / (1 + kq[1][1]) out[2][2] := kq[2][2] / (1 + kq[2][2]) out[3][3] := kq[3][3] / (1 + kq[3][3]) others ud.	kq	out[1][1] := 1 - sqrt(kq[1][1]) out[2][2] := 1 - sqrt(kq[2][2]) out[3][3] := 1 - sqrt(kq[3][3]) others ud	out[1][1] := 1 - 1 / sqrt(kq[1][1]) out[2][2] := 1 - 1 / sqrt(kq[2][2]) out[3][3] := 1 - 1 / sqrt(kq[3][3]) others ud.

G

Table D-33 out for eist_g_knownquantity

H₁ ∧ H₂ → G

Continue→

$H_2 : \text{kgflag}$

H_1

	lfst	efst	fist	eist	tst
	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$	$\text{Out} := [\text{ud}]$	kq	$\text{Out} := [\text{ud}]$
$\text{kq}[1][1] = \text{ud} \wedge \text{kq}[2][2] = \text{ud} \wedge \text{kq}[3][3] = \text{ud}$	$\text{out}[3][3] := 1 - 1/\sqrt{2 * \text{kq}[3][3] + 1}$ others ud.	$\text{out}[3][3] := 1 - \sqrt{1 - 2 * \text{kq}[3][3]}$ others ud	$\text{out}[3][3] := \text{kq}[3][3]/(1 + \text{kq}[3][3])$ others ud.	kq	$\text{out}[3][3] := 1 - \exp(-\text{kq}[3][3])$ others ud.
$\text{kq}[1][1] = \text{ud} \wedge \text{kq}[2][2] \neq \text{ud} \wedge \text{kq}[3][3] = \text{ud}$	$\text{out}[2][2] := 1 - 1/\sqrt{2 * \text{kq}[2][2] + 1}$ others ud.	$\text{out}[2][2] := 1 - \sqrt{1 - 2 * \text{kq}[2][2]}$ others ud	$\text{out}[2][2] := \text{kq}[2][2]/(1 + \text{kq}[2][2])$ others ud.	kq	$\text{out}[2][2] := 1 - \exp(-\text{kq}[2][2])$ others ud.
$\text{kq}[1][1] = \text{ud} \wedge \text{kq}[2][2] \neq \text{ud} \wedge \text{kq}[3][3] \neq \text{ud}$	$\text{out}[2][2] := 1 - 1/\sqrt{2 * \text{kq}[2][2] + 1}$ $\text{out}[3][3] := 1 - 1/\sqrt{2 * \text{kq}[3][3] + 1}$ others ud.	$\text{out}[2][2] := 1 - \sqrt{1 - 2 * \text{kq}[2][2]}$ $\text{out}[3][3] := 1 - \sqrt{1 - 2 * \text{kq}[3][3]}$ others ud	$\text{out}[2][2] := \text{kq}[2][2]/(1 + \text{kq}[2][2])$ $\text{out}[3][3] := \text{kq}[3][3]/(1 + \text{kq}[3][3])$ others ud.	kq	$\text{out}[2][2] := 1 - \exp(-\text{kq}[2][2])$ $\text{out}[3][3] := 1 - \exp(-\text{kq}[3][3])$ others ud.
$\text{kq}[1][1] \neq \text{ud} \wedge \text{kq}[2][2] = \text{ud} \wedge \text{kq}[3][3] = \text{ud}$	$\text{out}[1][1] := 1 - 1/\sqrt{2 * \text{kq}[1][1] + 1}$ others ud.	$\text{out}[1][1] := 1 - \sqrt{1 - 2 * \text{kq}[1][1]}$ others ud	$\text{out}[1][1] := \text{kq}[1][1]/(1 + \text{kq}[1][1])$ others ud.	kq	$\text{out}[1][1] := 1 - \exp(-\text{kq}[1][1])$ others ud.
$\text{kq}[1][1] \neq \text{ud} \wedge \text{kq}[2][2] = \text{ud} \wedge \text{kq}[3][3] \neq \text{ud}$	$\text{out}[1][1] := 1 - 1/\sqrt{2 * \text{kq}[1][1] + 1}$ $\text{out}[3][3] := 1 - 1/\sqrt{2 * \text{kq}[3][3] + 1}$ others ud.	$\text{out}[1][1] := 1 - \sqrt{1 - 2 * \text{kq}[1][1]}$ $\text{out}[3][3] := 1 - \sqrt{1 - 2 * \text{kq}[3][3]}$ others ud	$\text{out}[1][1] := \text{kq}[1][1]/(1 + \text{kq}[1][1])$ $\text{out}[3][3] := \text{kq}[3][3]/(1 + \text{kq}[3][3])$ others ud.	kq	$\text{out}[1][1] := 1 - \exp(-\text{kq}[1][1])$ $\text{out}[3][3] := 1 - \exp(-\text{kq}[3][3])$ others ud.
$\text{kq}[1][1] \neq \text{ud} \wedge \text{kq}[2][2] \neq \text{ud} \wedge \text{kq}[3][3] = \text{ud}$	$\text{out}[1][1] := 1 - 1/\sqrt{2 * \text{kq}[1][1] + 1}$ $\text{out}[2][2] := 1 - 1/\sqrt{2 * \text{kq}[2][2] + 1}$ others ud.	$\text{out}[1][1] := 1 - \sqrt{1 - 2 * \text{kq}[1][1]}$ $\text{out}[2][2] := 1 - \sqrt{1 - 2 * \text{kq}[2][2]}$ others ud	$\text{out}[1][1] := \text{kq}[1][1]/(1 + \text{kq}[1][1])$ $\text{out}[2][2] := \text{kq}[2][2]/(1 + \text{kq}[2][2])$ others ud.	kq	$\text{out}[1][1] := 1 - \exp(-\text{kq}[1][1])$ $\text{out}[2][2] := 1 - \exp(-\text{kq}[2][2])$ others ud
$\text{kq}[1][1] \neq \text{ud} \wedge \text{kq}[2][2] \neq \text{ud} \wedge \text{kq}[3][3] \neq \text{ud}$	$\text{out}[1][1] := 1 - 1/\sqrt{2 * \text{kq}[1][1] + 1}$ $\text{out}[2][2] := 1 - 1/\sqrt{2 * \text{kq}[2][2] + 1}$ $\text{out}[3][3] := 1 - 1/\sqrt{2 * \text{kq}[3][3] + 1}$ others ud.	$\text{out}[1][1] := 1 - \sqrt{1 - 2 * \text{kq}[1][1]}$ $\text{out}[2][2] := 1 - \sqrt{1 - 2 * \text{kq}[2][2]}$ $\text{out}[3][3] := 1 - \sqrt{1 - 2 * \text{kq}[3][3]}$ others ud	$\text{out}[1][1] := \text{kq}[1][1]/(1 + \text{kq}[1][1])$ $\text{out}[2][2] := \text{kq}[2][2]/(1 + \text{kq}[2][2])$ $\text{out}[3][3] := \text{kq}[3][3]/(1 + \text{kq}[3][3])$ others ud.	kq	$\text{out}[1][1] := 1 - \exp(-\text{kq}[1][1])$ $\text{out}[2][2] := 1 - \exp(-\text{kq}[2][2])$ $\text{out}[3][3] := 1 - \exp(-\text{kq}[3][3])$ others ud

→Continue

G

Table D-33 out for eist_g_knownquantity (Continue)

$H_1 \wedge H_2 \rightarrow G$

H

disp.dispu=ud^ disp.dispv=ud^ disp.dispw=ud	Out :=[ud]
disp.dispu=ud^ disp.dispv=ud^ disp.dispw≠ud	$out[3][3] := disp.dispw / (disp.dispw + sg.width)$ others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw=ud	$out[2][2] := disp.dispv / (dispv + sg.height)$ others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw≠ud	$out[2][2] := disp.dispv / (disp.dispv + sg.height)$ $out[3][3] := disp.dispw / (disp.dispw + sg.width)$ others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw=ud	$out[1][1] := disp.dispu / (disp.dispu + sg.length)$ others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw≠ud	$out[1][1] := disp.dispu / (disp.dispu + sg.length)$ $out[3][3] := disp.dispw / (disp.dispw + sg.width)$ others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw=ud	$out[1][1] := disp.dispu / (disp.dispu + sg.length)$ $out[2][2] := disp.dispv / (disp.dispv + sg.height)$ others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw≠ud	$out[1][1] := disp.dispu / (disp.dispu + sg.length)$ $out[2][2] := disp.dispv / (disp.dispv + sg.height)$ $out[3][3] := disp.dispw / (disp.dispw + sg.width)$ others ud

Table D-34 out for eist_g_geometry

D. 3.25 True strain tensor module

Prefix: tst_

Reference: MG C. 3.2.21

D. 3.25.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

TensorFlagT from the tensor data definition module

KinematicsT from the table structure module

Imported constant:

DIM 3

Exported functions:

Routines name	Inputs	Outputs	Exception
tst_g_knownquantity	TensorDataT TensorFlagT	TensorDataT	
tst_g_geometry	SpecimenGeometryT DisplacementT	TensorDataT	none

External functions:

real lnfunctn(r: real)

D. 3.25.2 Interface semantics

State variable: none

Local variable:

temp: Real*[DIM][DIM]

State invariant: none

Assumption:

Access routine semantics:

tst_g_knownquantity(kq: TensorDataT, kqflag: TensorFlagT)

Exception: Table D-35

Output: Table D-36

tst_g_geometry(sg: SpecimenGeometryT, disp:DisplacementT)

Exception: none

Output: Table D-37

H_1	mdg	sdg	mdpg	sdpg	cdt	gdt	lfst
	$(kq[1][1] \leq 0) \vee$ $(kq[2][2] \leq 0) \vee$ $(kq[3][3] \leq 0)$ \Rightarrow ln_err	$(kq[1][1] = 0) \vee$ $(kq[2][2] = 0) \vee$ $(kq[3][3] = 0)$ \Rightarrow d_zero, $(kq[1][1] < 0) \vee$ $(kq[2][2] < 0) \vee$ $(kq[3][3] < 0)$ \Rightarrow ln_err	$(kq[1][1] \leq -1) \vee$ $(kq[2][2] \leq -1) \vee$ $(kq[3][3] \leq -1)$ \Rightarrow ln_err	$(kq[1][1] = 1) \vee$ $(kq[2][2] = 1) \vee$ $(kq[3][3] = 1)$ \Rightarrow d_zero, $(kq[1][1] > 1) \vee$ $(kq[2][2] > 1) \vee$ $(kq[3][3] > 1)$ \Rightarrow ln_err	$(kq[1][1] = 0) \vee$ $(kq[2][2] = 0) \vee$ $(kq[3][3] = 0)$ \Rightarrow d_zero, $(kq[1][1] < 0) \vee$ $(kq[2][2] < 0) \vee$ $(kq[3][3] < 0)$ \Rightarrow ln_err, sr_lesszero	$(kq[1][1] = 0) \vee$ $(kq[2][2] = 0) \vee$ $(kq[3][3] = 0)$ \Rightarrow ln_err, $(kq[1][1] < 0) \vee$ $(kq[2][2] < 0) \vee$ $(kq[3][3] < 0)$ \Rightarrow sr_lesszero	$(kq[1][1] = -0.5) \vee$ $(kq[2][2] = -0.5) \vee$ $(kq[3][3] = -0.5)$ \Rightarrow ln_err, $(kq[1][1] < -0.5) \vee$ $(kq[2][2] < -0.5) \vee$ $(kq[3][3] < -0.5)$ \Rightarrow sr_lesszero, ln_err
G							

$H_1 \rightarrow G$

Table D-35 exception for tst_g_knownquantity

continue \rightarrow

H_1	efst	list	eist	tst
	$(kq[1][1] = 0.5) \vee$ $(kq[2][2] = 0.5) \vee$ $(kq[3][3] = 0.5)$ \Rightarrow d_zero, $(kq[1][1] > 0.5) \vee$ $(kq[2][2] > 0.5) \vee$ $(kq[3][3] > 0.5)$ \Rightarrow sr_lesszero, ln_err	$(kq[1][1] \leq -1) \vee$ $(kq[2][2] \leq -1) \vee$ $(kq[3][3] \leq -1)$ \Rightarrow ln_err	$(kq[1][1] = 1) \vee$ $(kq[2][2] = 1) \vee$ $(kq[3][3] = 1)$ \Rightarrow d_zero, $(kq[1][1] > 1) \vee$ $(kq[2][2] > 1) \vee$ $(kq[3][3] > 1)$ \Rightarrow ln_err	non
G				

$H_1 \rightarrow G$

continue \rightarrow Table D-35 exception for tst_g_knownquantity

H₁

	mdg	sdg	mdpg	sdpg	cdt
	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]	Out:=[ud]
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]=ud					
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[3][3] := ln(kq[3][3]) others ud	out[3][3] := ln(1/kq[3][3]) others ud	out[3][3] := ln(1+kq[3][3]) others ud.	out[3][3] := ln(1/(1-kq[3][3])) others ud	out[3][3] := ln(1/sqrt(kq[3][3])) others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[2][2] := ln(kq[2][2]) others ud	out[2][2] := ln(1/kq[2][2]) others ud	out[2][2] := ln(1+kq[2][2]) others ud.	out[2][2] := ln(1/(1-kq[2][2])) others ud	out[2][2] := ln(1/sqrt(kq[2][2])) others ud.
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[2][2] := ln(kq[2][2]) out[3][3] := ln(kq[3][3]) others ud	out[2][2] := ln(1/kq[2][2]) out[3][3] := ln(1/kq[3][3]) others ud	out[2][2] := ln(1+kq[2][2]) out[3][3] := ln(1+kq[3][3]) others ud	out[2][2] := ln(1/(1-kq[2][2])) out[3][3] := ln(1/(1-kq[3][3])) others ud	out[2][2] := ln(1/sqrt(kq[2][2])) others ud
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]=ud	out[1][1] := ln(kq[1][1]) others ud	out[1][1] := ln(1/kq[1][1]) others ud	out[1][1] := ln(1+kq[1][1]) others ud.	out[1][1] := ln(1/(1-kq[1][1])) others ud	out[1][1] := ln(1/sqrt(kq[1][1])) others ud.
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[1][1] := ln(kq[1][1]) out[3][3] := ln(kq[3][3]) others ud	out[1][1] := ln(1/kq[1][1]) out[3][3] := ln(1/kq[3][3]) others ud	out[1][1] := ln(1+kq[1][1]) out[3][3] := ln(1+kq[3][3]) others ud	out[1][1] := ln(1/(1-kq[1][1])) out[3][3] := ln(1/(1-kq[3][3])) others ud	out[1][1] := ln(1/sqrt(kq[1][1])) out[3][3] := ln(1/sqrt(kq[3][3])) others ud
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[1][1] := ln(kq[1][1]) out[2][2] := ln(kq[2][2]) others ud	out[1][1] := ln(1/kq[1][1]) out[2][2] := ln(1/kq[2][2]) others ud	out[1][1] := ln(1+kq[1][1]) out[2][2] := ln(1+kq[2][2]) others ud	out[1][1] := ln(1/(1-kq[1][1])) out[2][2] := ln(1/(1-kq[2][2])) others ud	out[1][1] := ln(1/sqrt(kq[1][1])) out[2][2] := ln(1/sqrt(kq[2][2])) others ud
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[1][1] := ln(kq[1][1]) out[2][2] := ln(kq[2][2]) out[3][3] := ln(kq[3][3]) others ud	out[1][1] := ln(1/kq[1][1]) out[2][2] := ln(1/kq[2][2]) out[3][3] := ln(1/kq[3][3]) others ud	out[1][1] := ln(1+kq[1][1]) out[2][2] := ln(1+kq[2][2]) out[3][3] := ln(1+kq[3][3]) others ud	out[1][1] := ln(1/(1-kq[1][1])) out[2][2] := ln(1/(1-kq[2][2])) out[3][3] := ln(1/(1-kq[3][3])) others ud	out[1][1] := ln(1/sqrt(kq[1][1])) out[2][2] := ln(1/sqrt(kq[2][2])) out[3][3] := ln(1/sqrt(kq[3][3])) others ud

G

Table D-36 out for tst_g_knownquantity

H₁ ∧ H₂ → G

Continue→

H_1

	gdt	lfst	efst
$kq[1][1]=ud \wedge kq[2][2]=ud \wedge kq[3][3]=ud$	Out :=[ud]	Out:=[ud]	Out:=[ud]
$kq[1][1]=ud \wedge kq[2][2]=ud \wedge kq[3][3] \neq ud$	$out[3][3] := \ln(\text{sqrt}(kq[3][3]))$ others ud	$out[3][3] := \ln(\text{sqrt}(2 * kq[3][3] + 1))$ others ud	$out[3][3] := \ln(1 / \text{sqrt}(1 - 2 * kq[3][3]))$ others ud
$kq[1][1]=ud \wedge kq[2][2] \neq ud \wedge kq[3][3]=ud$	$out[2][2] := \ln(\text{sqrt}(kq[2][2]))$ others ud	$out[2][2] := \ln(\text{sqrt}(2 * kq[2][2] + 1))$ others ud	$out[2][2] := \ln(1 / \text{sqrt}(1 - 2 * kq[2][2]))$ others ud
$kq[1][1]=ud \wedge kq[2][2] \neq ud \wedge kq[3][3] \neq ud$	$out[2][2] := \ln(\text{sqrt}(kq[2][2]))$ $out[3][3] := \ln(\text{sqrt}(kq[3][3]))$ others ud	$out[2][2] := \ln(\text{sqrt}(2 * kq[2][2] + 1))$ $out[3][3] := \ln(\text{sqrt}(2 * kq[3][3] + 1))$ others ud	$out[2][2] := \ln(1 / \text{sqrt}(1 - 2 * kq[2][2]))$ $out[3][3] := \ln(1 / \text{sqrt}(1 - 2 * kq[3][3]))$ others ud
$kq[1][1] \neq ud \wedge kq[2][2]=ud \wedge kq[3][3]=ud$	$out[1][1] := \ln(\text{sqrt}(kq[1][1]))$ others ud	$out[1][1] := \ln(\text{sqrt}(2 * kq[1][1] + 1))$ others ud	$out[1][1] := \ln(1 / \text{sqrt}(1 - 2 * kq[1][1]))$ others ud
$kq[1][1] \neq ud \wedge kq[2][2]=ud \wedge kq[3][3] \neq ud$	$out[1][1] := \ln(\text{sqrt}(kq[1][1]))$ $out[3][3] := \ln(\text{sqrt}(kq[3][3]))$ others ud	$out[1][1] := \ln(\text{sqrt}(2 * kq[1][1] + 1))$ $out[3][3] := \ln(\text{sqrt}(2 * kq[3][3] + 1))$ others ud	$out[1][1] := \ln(1 / \text{sqrt}(1 - 2 * kq[1][1]))$ $out[3][3] := \ln(1 / \text{sqrt}(1 - 2 * kq[3][3]))$ others ud
$kq[1][1] \neq ud \wedge kq[2][2] \neq ud \wedge kq[3][3]=ud$	$out[1][1] := \ln(\text{sqrt}(kq[1][1]))$ $out[2][2] := \ln(\text{sqrt}(kq[2][2]))$ others ud	$out[1][1] := \ln(\text{sqrt}(2 * kq[1][1] + 1))$ $out[2][2] := \ln(\text{sqrt}(2 * kq[2][2] + 1))$ others ud	$out[1][1] := \ln(1 / \text{sqrt}(1 - 2 * kq[1][1]))$ $out[2][2] := \ln(1 / \text{sqrt}(1 - 2 * kq[2][2]))$ others ud
$kq[1][1] \neq ud \wedge kq[2][2] \neq ud \wedge kq[3][3] \neq ud$	$out[1][1] := \ln(\text{sqrt}(kq[1][1]))$ $out[2][2] := \ln(\text{sqrt}(kq[2][2]))$ $out[3][3] := \ln(\text{sqrt}(kq[3][3]))$ others ud	$out[1][1] := \ln(\text{sqrt}(2 * kq[1][1] + 1))$ $out[2][2] := \ln(\text{sqrt}(2 * kq[2][2] + 1))$ $out[3][3] := \ln(\text{sqrt}(2 * kq[3][3] + 1))$ others ud	$out[1][1] := \ln(1 / \text{sqrt}(1 - 2 * kq[1][1]))$ $out[2][2] := \ln(1 / \text{sqrt}(1 - 2 * kq[2][2]))$ $out[3][3] := \ln(1 / \text{sqrt}(1 - 2 * kq[3][3]))$ others ud

→Continue G

Table D-36 out for tst_g_knownquantity (Continue)

$H_1 \wedge H_2 \rightarrow G$

H₂ : kgflag

H₁

	list	eist	tst
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]=ud	Out:=[ud]	Out:=[ud]	kq
kq[1][1]=ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[3][3] := ln(1 + kq[3][3]) others ud.	out[3][3] := ln(1/(1 - kq[3][3])) others ud	kq
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[2][2] := ln(1 + kq[2][2]) others ud.	out[2][2] := ln(1/(1 - kq[2][2])) others ud	kq
kq[1][1]=ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[2][2] := ln(1 + kq[2][2]) out[3][3] := ln(1 + kq[3][3]) others ud	out[2][2] := ln(1/(1 - kq[2][2])) out[3][3] := ln(1/(1 - kq[3][3])) others ud	kq
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]=ud	out[1][1] := ln(1 + kq[1][1]) others ud.	out[1][1] := ln(1/(1 - kq[1][1])) others ud	kq
kq[1][1]≠ud^ kq[2][2]=ud^ kq[3][3]≠ud	out[1][1] := ln(1 + kq[1][1]) out[3][3] := ln(1 + kq[3][3]) others ud	out[1][1] := ln(1/(1 - kq[1][1])) out[3][3] := ln(1/(1 - kq[3][3])) others ud	kq
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]=ud	out[1][1] := ln(1 + kq[1][1]) out[2][2] := ln(1 + kq[2][2]) others ud	out[1][1] := ln(1/(1 - kq[1][1])) out[2][2] := ln(1/(1 - kq[2][2])) others ud	kq
kq[1][1]≠ud^ kq[2][2]≠ud^ kq[3][3]≠ud	out[1][1] := ln(1 + kq[1][1]) out[2][2] := ln(1 + kq[2][2]) out[3][3] := ln(1 + kq[3][3]) others ud	out[1][1] := ln(1/(1 - kq[1][1])) out[2][2] := ln(1/(1 - kq[2][2])) out[3][3] := ln(1/(1 - kq[3][3])) others ud	kq

H₁ ∧ H₂ → G

→Continue G Table D-36 out for tst_g_knownquantity (Continue)

H

disp.dispu=ud^ disp.dispv=ud^ disp.dispw=ud	Out :=[ud]
disp.dispu=ud^ disp.dispv=ud^ disp.dispw≠ud	$out[3][3] := \ln(1 + dispw / sg.width)$ others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw=ud	$out[2][2] := \ln(1 + disp.dispv / sg.height)$ others ud
disp.dispu=ud^ disp.dispv≠ud^ disp.dispw≠ud	$out[2][2] := \ln(1 + disp.dispv / sg.height)$ $out[3][3] := \ln(1 + dispw / sg.width)$ others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw=ud	$out[1][1] := \ln(1 + disp.dispu / sg.length)$ others ud
disp.dispu≠ud^ disp.dispv=ud^ disp.dispw≠ud	$out[1][1] := \ln(1 + disp.dispu / sg.length)$ $out[3][3] := \ln(1 + dispw / sg.width)$ others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw=ud	$out[1][1] := \ln(1 + disp.dispu / sg.length)$ $out[2][2] := \ln(1 + disp.dispv / sg.height)$ others ud
disp.dispu≠ud^ disp.dispv≠ud^ disp.dispw≠ud	$out[1][1] := \ln(1 + disp.dispu / sg.length)$ $out[2][2] := \ln(1 + disp.dispv / sg.height)$ $out[3][3] := \ln(1 + dispw / sg.width)$ others ud

Table D-37 out for tst_g_geometry

D. 3.26 Displacement constitutive calculation module

Prefix: dcc_

Reference: MG C. 3.2.7

D. 3.26.1 Interface syntax

Imported data type:

DisplacementT from the displacement specification module

KinematicsT from the table structure module

StressT from the table structure module

TensorDataT from the material deformation gradient module

ExperimentClassT from the experiment definition module

ExperimentTypeT from the experiment definition module

PropertylistT from material properties file module

Exported data type: none

Exported constant: none

External functions:

All access routines in this module interface specification are available.

Exported function:

Routines name	Inputs	Outputs	Exception
Virtual dcc_dispconstitutive	DisplacementT KinematicsT SpecimenGeometryT PropertylistT	StressT KinematicsT	

D. 3.26.2 Interface semantics

State variable: none

State invariant: none

Assumption:

- The experiment designer can understand module interface specification.
- When the constitutive equation is involved in the experiment, since it is hard to summarize the constitutive equation as a unique form with a fixed pattern, the experiment designer is given more freedom to provide the implementation of this routine and also the experiment designer can also design the local functions based on the requirements except that the designer can call all access routines in the module interface specification

Access routine semantics:

virtual dcc_dispconstitutive(disp: DisplacementT, spe: SpecimenGeometryT, prolist: PropertylistT, kin: KinematicsT)

exception: Exception occurs depending on known kinematics quantities and how to calculate the approximation of the constitutive equation. The experiment designer is responsible for exceptions in this routine.

output:

Stresses are obtained based on the approximation of the constitutive equation and known conditions such as displacements and kinematics quantities. Based on the constitutive equation, kinematics quantities can be recalculated again.

D. 3.27 Load constitutive calculation module

Prefix: lcc_

Reference: MG C. 3.2.8

D. 3.27.1 Interface syntax

Imported data type:

- LoadT from the load specification module
- KinematicsT from the table structure module
- StressT from the table structure module
- TensorDataT from the tensor data definition module
- ExperimentClassT from the experiment definition module
- ExperimentTypeT from the experiment definition module
- PropertylistT from material properties file module

Exported data type: none

Exported constant: none

Exported function:

Routines name	Inputs	Outputs	Exception
virtual lcc_loadconstitutive	LoadT StressT KinematicsT SpecimenGeometryT PropertylistT	StressT KinematicsT DisplacementT	

D. 3.27.2 Interface semantics

State variable: none

State invariant: none

Assumption:

- The experiment designer can understand the module interface specification.
- When the constitutive equation is involved in the experiment, since it is hard to summarize the constitutive equation as a unique form with a fixed pattern, the experiment designer is given more freedom to provide the implementation of this routine and also the experiment designer can also design the local functions based on the requirements.

Access routine semantics:

virtual lcc_loadconstitutive(load: LoadT, stress: StressT, spe: SpecimenGeometryT, prolist: PropertylistT, kin: KinematicsT)

exception: Exception occurs depending on known kinematics quantities load and how to calculate the approximation of the constitutive equation. The experiment designer is responsible for exceptions in this routine.

output: That which stress and which strain are used in the constitutive equation decides how to calculate the approximation of constitutive equation. Displacements can be obtained by the strain. Once the displacement is known, other kinematics quantities can be calculated.

D. 3.28 Engineering stress module

Prefix: es_

Reference: MG C. 3.2.23

D. 3.28.1 Interface syntax

Imported data type:

SpecimenGeometryT from the specimen geometry module
 DisplacementT from the displacement specification module
 LoadT from the load specification module
 TensorDataT from the tensor data definition module

Imported constant:

DIM 3

Exported constant: none

Exported functions:

Routines name	Inputs	Outputs	Exception
es_initengstress		TensorDataT	none
es_g_engstress	LoadT SpecimenGeometryT	TensorDataT	none
es_g_knownstress	TensorDataT DisplacementT SpecimenGeometryT	TensorDataT	none

External function:

ed_g_experimenttype

D. 3.28.2 Interface semantic

State variable: none

State invariant: none

Assumption:

- es_initengstress should be called before other access routines.
- Before es_knownstress is called, displacement deformation should be obtained by calling lcc_loadconstitutive access routine.

Access routine semantics:

Es_initengstress()

Exception: none.

Output: Table D-38

es_g_engstress(load: LoadT, spe: SpecimenGeometryT,
 truestress:TensorDataT)

exception: none.

output: Table D-39

es_g_knownstress(displ: DisplacementT, spe: SpecimenGeometryT, truestress: TensorDataT)
 exception: none

output: Table D-40

(localtype=H)→G

H	Uniaxial	Biaxial	Multiaxial	
G	$\begin{bmatrix} 0 & ud & ud \\ ud & ud & ud \\ ud & ud & ud \end{bmatrix}$	$\begin{bmatrix} 0 & ud & ud \\ ud & 0 & ud \\ ud & ud & ud \end{bmatrix}$	$\begin{bmatrix} 0 & ud & ud \\ ud & 0 & ud \\ ud & ud & 0 \end{bmatrix}$	H→G

Table D-38 out for es_initengstress

G

H		Out :=[ud]	
	load.loadu=ud^ load.loadv=ud^ load.loadw=ud		H→G
	load.loadu=ud^ load.loadv=ud^ load.loadw≠ud	out[3][3] := load.loadw/(spe.length * spe.height) others ud	
	load.loadu=ud^ load.loadv≠ud^ load.loadw=ud	out[2][2] := load.loadv/(spe.length * spe.width) others ud	
	load.loadu=ud^ load.loadv≠ud^ load.loadw≠ud	out[2][2] := load.loadv/(spe.length * spe.width) out[3][3] := load.loadw/(spe.length * spe.height) others ud	
	load.loadu≠ud^ load.loadv=ud^ load.loadw=ud	out[1][1] := load.loadu/(spe.height * spe.width) others ud	
	load.loadu≠ud^ load.loadv=ud^ load.loadw≠ud	out[1][1] := load.loadu/(spe.height * spe.width) out[3][3] := load.loadw/(spe.length * spe.height) others ud	
	load.loadu≠ud^ load.loadv≠ud^ load.loadw=ud	out[1][1] := load.loadu/(spe.height * spe.width) out[2][2] := load.loadv/(spe.length * spe.width) others ud	
	load.loadu≠ud^ load.loadv≠ud^ load.loadw≠ud	out[1][1] := load.loadu/(spe.height * spe.width) out[2][2] := load.loadv/(spe.length * spe.width) out[3][3] := load.loadw/(spe.length * spe.height) others ud	

Table D-39 out for es_g_engstress

G

H

H→G

ts[1][1]=ud^ ts[2][2]=ud^ ts[3][3]=ud	Out:={ud}
ts[1][1]=ud^ ts[2][2]=ud^ ts[3][3]≠ud	$out[3][3] := ts[3][3] \frac{(spe.length + disp.dispu)(spe.height + disp.dispv)}{spe.length * spe.height}$ others ud
ts[1][1]=ud^ ts[2][2]≠ud^ ts[3][3]=ud	$out[2][2] := ts[2][2] \frac{(spe.length + disp.dispu)(spe.width + disp.dispw)}{spe.length * spe.width}$ others ud
ts[1][1]=ud^ ts[2][2]≠ud^ ts[3][3]≠ud	$out[2][2] := ts[2][2] \frac{(spe.length + disp.dispu)(spe.width + disp.dispw)}{spe.length * spe.width}$ $out[3][3] := ts[3][3] \frac{(spe.length + disp.dispu)(spe.height + disp.dispv)}{spe.length * spe.height}$ others ud
ts[1][1]≠ud^ ts[2][2]=ud^ ts[3][3]=ud	$out[1][1] := ts[1][1] \frac{(spe.width + disp.dispw)(spe.height + disp.dispv)}{spe.width * spe.height}$ others ud
ts[1][1]≠ud^ ts[2][2]=ud^ ts[3][3]≠ud	$out[1][1] := ts[1][1] \frac{(spe.width + disp.dispw)(spe.height + disp.dispv)}{spe.width * spe.height}$ $out[3][3] := ts[3][3] \frac{(spe.length + disp.dispu)(spe.height + disp.dispv)}{spe.length * spe.height}$ others ud
ts[1][1]≠ud^ ts[2][2]≠ud^ ts[3][3]=ud	$out[1][1] := ts[1][1] \frac{(spe.width + disp.dispw)(spe.height + disp.dispv)}{spe.width * spe.height}$ $out[2][2] := ts[2][2] \frac{(spe.length + disp.dispu)(spe.width + disp.dispw)}{spe.length * spe.width}$ others ud
ts[1][1]≠ud^ ts[2][2]≠ud^ ts[3][3]≠ud	$out[1][1] := ts[1][1] \frac{(spe.width + disp.dispw)(spe.height + disp.dispv)}{spe.width * spe.height}$ $out[2][2] := ts[2][2] \frac{(spe.length + disp.dispu)(spe.width + disp.dispw)}{spe.length * spe.width}$ $out[3][3] := ts[3][3] \frac{(spe.length + disp.dispu)(spe.height + disp.dispv)}{spe.length * spe.height}$ others ud

Table D-40 out for es_g_knownstress

D. 3.29 True stress module

Prefix: tsm_

Reference: MG C. 3.2.22

D. 3.29.1 Interface syntax

Imported data type:

SpecimenGeometryT from the specimen geometry module

LoadT from the load specification module

DisplacementT from the displacement specification module

TensorDataT from the tensor data definition module

Exported constant: none

Exported functions:

Routines name	Inputs	Outputs	Exception
tsm_inittruestress		TensorDataT	none
virtual tsm_g_truestress	LoadT KinematicsT SpecimenGeometryT	TensorDataT	
tsm_g_knownstress	TensorDataT DisplacementT SpecimenGeometryT	TensorDataT	none

D. 3.29.2 Interface semantics

State variable: none

State invariant: none

Assumption:

- tsm_inittruestress should be called to initialize true stress before other access routines.
- Before tsm_g_knownstress is called, displacement deformation should be obtained by calling lcc_loadconstitutive access routine.
- True stress is needed in the constitutive equation. If engineering stress is needed in the corresponding constitutive equation, then tsm_g_truestress is meaningless.

Access routine semantics:

tsm_inittruestress()

Exception: none.

Output: Table D-41

virtual tsm_g_truestress(load: LoadT, spe: SpecimenGeometryT, kin: KinematicsT)

exception: Exception occurs depending on how to calculate the approximation of the constitutive equation. The experiment designer is responsible for exceptions in this routine.

output: The experiment designer is responsible for the implementation of this routine.

tsm_g_knownstress(displ: DisplacementT, spe: SpecimenGeometryT,
engstress:TensorDataT)

exception: none.

Output: Table D-42

(localtype=H)→G

H	Uniaxial	Biaxial	Multiaxial	
G	$\begin{bmatrix} 0 & ud & ud \\ ud & ud & ud \\ ud & ud & ud \end{bmatrix}$	$\begin{bmatrix} 0 & ud & ud \\ ud & 0 & ud \\ ud & ud & ud \end{bmatrix}$	$\begin{bmatrix} 0 & ud & ud \\ ud & 0 & ud \\ ud & ud & 0 \end{bmatrix}$	H→G

Table D-41 out for tsm_initruestress

G

H	es[1][1]=ud^ es[2][2]=ud^ es[3][3]=ud	Out:=[ud]	
	es[1][1]=ud^ es[2][2]=ud^ es[3][3]≠ud	$out[3][3] := es[3][3] \frac{spe.length * spe.height}{(disp.dispu + spe.length)(spe.height + disp.dispv)}$ others ud	H→G
	es[1][1]=ud^ es[2][2]≠ud^ es[3][3]=ud	$out[2][2] := es[2][2] \frac{spe.length * spe.width}{(spe.width + disp.dispw)(spe.length + disp.dispu)}$ others ud	
	es[1][1]=ud^ es[2][2]≠ud^ es[3][3]≠ud	$out[2][2] := es[2][2] \frac{spe.length * spe.width}{(spe.width + disp.dispw)(spe.length + disp.dispu)}$ $out[3][3] := es[3][3] \frac{spe.length * spe.height}{(disp.dispu + spe.length)(spe.height + disp.dispv)}$ others ud	
	es[1][1]≠ud^ es[2][2]=ud^ es[3][3]=ud	$out[1][1] := es[1][1] \frac{spe.width * spe.height}{(disp.dispw + spe.width)(disp.dispv + spe.height)}$ others ud	
	es[1][1]≠ud^ es[2][2]=ud^ es[3][3]≠ud	$out[1][1] := es[1][1] \frac{spe.width * spe.height}{(disp.dispw + spe.width)(disp.dispv + spe.height)}$ $out[3][3] := es[3][3] \frac{spe.length * spe.height}{(disp.dispu + spe.length)(spe.height + disp.dispv)}$ others ud	
	es[1][1]≠ud^ es[2][2]≠ud^ es[3][3]=ud	$out[1][1] := es[1][1] \frac{spe.width * spe.height}{(disp.dispw + spe.width)(disp.dispv + spe.height)}$ $out[2][2] := es[2][2] \frac{spe.length * spe.width}{(spe.width + disp.dispw)(spe.length + disp.dispu)}$ others ud	
	es[1][1]≠ud^ es[2][2]≠ud^ es[3][3]≠ud	$out[1][1] := es[1][1] \frac{spe.width * spe.height}{(disp.dispw + spe.width)(disp.dispv + spe.height)}$ $out[2][2] := es[2][2] \frac{spe.length * spe.width}{(spe.width + disp.dispw)(spe.length + disp.dispu)}$ $out[3][3] := es[3][3] \frac{spe.length * spe.height}{(disp.dispu + spe.length)(spe.height + disp.dispv)}$ others ud	

Table D-42 out for tsm_g_knownstress

D. 3.30 Output show module

Prefix: os_

Reference: MG C. 3.1.4

D. 3.30.1 Interface syntax

Exported data type:

CoordDataT = sequence of Real*

Exported constant:

None

Exported functions:

Routines name	Inputs	Outputs	Exception
os_curveshow	CoordDataT CoordDataT		Undefined_data

D. 3.30.2 Interface semantics

State variable:

None

State invariant:

None

Assumption:

None

Access routine semantics:

os_curveshow(x: CoordDataT, y: CoordDataT)

exception: $(x[0]=undefined) \vee (y[0]=undefined) \Rightarrow \text{undefined_data}$

output: In the coordinate system all the points whose value (x,y) is respectively from the inputs x and y. The curve that is composed of all the points will be drawn in the coordinate system.

Appendix E Component description for Virlab

1 Structure component

Name	Structure component
Role in the system	Fundamental component
Service	Provide the data structures that are used to represent the required information by the experiment. These data structures act as a bridge between outside information input by the specification and inside information required by the algorithms. The structure component communicates with outside specifications and algorithms by its interfaces.
Composition	Constitutive equation (con_equ) structure module Displacement (disp) structure module Load (load) structure module Tensor structure module Experiment (exp) definition module Specimen (spe) geometry module
Interface specification	The structure component interfaces are composed with its component modules' interfaces. Please refer to the corresponding module interface specification found in the Appendix D: module interface specification for the Virlab

2 Stress component

Name	Stress component				
Role in the system	Functional component				
Service	Stress component is used for the calculation of the true stress and engineering stress				
Composition	True stress tensor module Engineering stress tensor module				
Interface specification	<p>1. The composed modules' interfaces are available for the use. Please refer to the module interface specification found in the appendix D</p> <p>2. Based on the similarities between the engineering stress module's interfaces and true stress module's interface, two additional interfaces are summarized below.</p> <ul style="list-style-type: none"> • sc_initstress(outflag: TensorFlagT) Output: output is based on the value of outflag shown in the table E-2-1 Exception: exceptions are triggered from the calling programs and same as the exceptions from the calling programs <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>Outflag = TSST</td> <td>tsm_inittruestress</td> </tr> <tr> <td>Outflag = ESST</td> <td>es_initengstress</td> </tr> </table> <p style="text-align: center;">Table E-3-1 output for sc_initstress</p>	Outflag = TSST	tsm_inittruestress	Outflag = ESST	es_initengstress
Outflag = TSST	tsm_inittruestress				
Outflag = ESST	es_initengstress				

	<ul style="list-style-type: none"> • <code>sc_knownstress(outflag:TensorFlagT, ss: TensorDataT; disp: DisplacementT; sg: SpecimenGeometryT)</code> Output: output is based on the value of outflag shown in the table E-2-2 Exception: exceptions are triggered from the calling programs and same as the exceptions from the calling programs <table border="1"> <tr> <td>Outflag = TSST</td> <td><code>Tsm_g_knownstress(ss,disp,sg)</code></td> </tr> <tr> <td>Outflag = ESST</td> <td><code>es_g_knownstress(ss,disp,sg)</code></td> </tr> </table> <p>Table E-2-2 output for <code>sc_knownstress</code></p>	Outflag = TSST	<code>Tsm_g_knownstress(ss,disp,sg)</code>	Outflag = ESST	<code>es_g_knownstress(ss,disp,sg)</code>
Outflag = TSST	<code>Tsm_g_knownstress(ss,disp,sg)</code>				
Outflag = ESST	<code>es_g_knownstress(ss,disp,sg)</code>				
Comments	<ol style="list-style-type: none"> TensorFlagT is from the Tensor data definition module in the structure component TensorDataT is from tensor data definition module in the structure component DisplacementT is from the displacement structure module in the structure component SpecimenGeometryT is from the specimen geometry module in the structure component Capital letters such as MDPG are from the tensor data definition module in the structure component. 				

3 `disp_con` component

Name	<code>disp_con</code> component
Role in the system	Functional component
Service	This component provides the algorithms to approximate the constitutive equation in the displacement-controlled experiment.
Composition	displacement constitutive (<code>disp_con</code>) calculation module
Interface specification	This component just includes one module and its interfaces are also the composed module's interfaces. Please refer to the module interface specification found in the appendix D

4 `load_con` component

Name	<code>load_con</code> component
Role in the system	Functional component
Service	This component provides the algorithms to approximate the constitutive equation in the load-controlled experiment.
Composition	load constitutive (<code>load_con</code>) calculation module
Interface specification	This component just includes one module and its interfaces are also the composed module's interfaces. Please refer to the module interface specification found in the appendix D

5 kinematics component

Name	kinematics component
------	----------------------

Role in the system	Functional component																						
Service	Based on the definitions given by [MG70], kinematics component is used to calculate the kinematics quantities.																						
Composition	<p>material deformation gradient module spatial deformation gradient module material displacement gradient module spatial displacement gradient module Cauchy's deformation tensor module Green's deformation tensor module Lagrangian finite strain tensor module Eulerian finite strain tensor module Lagrangian infinitesimal strain tensor module Eulerian infinitesimal strain tensor module True strain tensor module Stretch tensor module Stretch ratio tensor module</p>																						
Interface specification	<p>The composed modules' interfaces are available for the use. Please refer to the corresponding module interface specification found in the appendix D. Based on the similarities among all the composed modules' interfaces in the kinematics component, two interfaces are summarized below.</p> <p>a. <code>kc_knownquantity(comeflag, outflag: TensorFlagT; kq: TensorDataT)</code> Output: output is based on the value of outflag shown in the table E-5-1 Exception: exceptions are triggered from the calling programs and same as the exceptions from the calling programs</p> <table border="1" data-bbox="511 1216 1305 1645"> <tr><td>outflag=MDG</td><td><code>mdg_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=SDG</td><td><code>sdg_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=MDPG</td><td><code>mdpg_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=SDPG</td><td><code>sdpg_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=CDT</td><td><code>cdt_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=GDT</td><td><code>gdt_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=LFST</td><td><code>lfst_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=EFST</td><td><code>efst_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=LIST</td><td><code>list_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=EIST</td><td><code>eist_g_knownquantity(kq, comeflag)</code></td></tr> <tr><td>outflag=TST</td><td><code>tst_g_knownquantity(kq, comeflag)</code></td></tr> </table> <p>Table E-5-1: output for <code>kc_knownquantity</code></p> <p>b. <code>kc_geometry(outflag: TensorFlagT, disp: DisplacementT, sg: SpecimenGeometryT)</code> Output: output is based on the value of outflag shown in the table E-5-2 Exception: exceptions are triggered from the calling</p>	outflag=MDG	<code>mdg_g_knownquantity(kq, comeflag)</code>	outflag=SDG	<code>sdg_g_knownquantity(kq, comeflag)</code>	outflag=MDPG	<code>mdpg_g_knownquantity(kq, comeflag)</code>	outflag=SDPG	<code>sdpg_g_knownquantity(kq, comeflag)</code>	outflag=CDT	<code>cdt_g_knownquantity(kq, comeflag)</code>	outflag=GDT	<code>gdt_g_knownquantity(kq, comeflag)</code>	outflag=LFST	<code>lfst_g_knownquantity(kq, comeflag)</code>	outflag=EFST	<code>efst_g_knownquantity(kq, comeflag)</code>	outflag=LIST	<code>list_g_knownquantity(kq, comeflag)</code>	outflag=EIST	<code>eist_g_knownquantity(kq, comeflag)</code>	outflag=TST	<code>tst_g_knownquantity(kq, comeflag)</code>
outflag=MDG	<code>mdg_g_knownquantity(kq, comeflag)</code>																						
outflag=SDG	<code>sdg_g_knownquantity(kq, comeflag)</code>																						
outflag=MDPG	<code>mdpg_g_knownquantity(kq, comeflag)</code>																						
outflag=SDPG	<code>sdpg_g_knownquantity(kq, comeflag)</code>																						
outflag=CDT	<code>cdt_g_knownquantity(kq, comeflag)</code>																						
outflag=GDT	<code>gdt_g_knownquantity(kq, comeflag)</code>																						
outflag=LFST	<code>lfst_g_knownquantity(kq, comeflag)</code>																						
outflag=EFST	<code>efst_g_knownquantity(kq, comeflag)</code>																						
outflag=LIST	<code>list_g_knownquantity(kq, comeflag)</code>																						
outflag=EIST	<code>eist_g_knownquantity(kq, comeflag)</code>																						
outflag=TST	<code>tst_g_knownquantity(kq, comeflag)</code>																						

programs	
outflag=MDG	mdg g geometry(sg,disp)
outflag=SDG	sdg g geometry(sg,disp)
outflag=MDPG	mdpg g geometry(sg,disp)
outflag=SDPG	sdp g geometry(sg,disp)
outflag=CDT	cdt g geometry(sg,disp)
outflag=GDT	gdt g geometry(sg,disp)
outflag=LFST	lfst g geometry(sg,disp)
outflag=EFST	efst g geometry(sg,disp)
outflag=LIST	list g geometry(sg,disp)
outflag=EIST	eist g geometry(sg,disp)
outflag=TST	tst g geometry(sg,disp)

Table E-5-2: output for kc geometry

Comments	<ul style="list-style-type: none"> c. TensorFlagT is from the Tensor data definition module in the structure component d. TensorDataT is from tensor data definition module in the structure component e. DisplacementT is from the displacement structure module in the structure component f. SpecimenGeometryT is from the specimen geometry module in the structure component g. Capital letters such as MDPG are from the tensor data definition module in the structure component.
----------	---

6 table component

Name	Table component
Role in the system	Application component
Service	Table component is specially designed to describe the experimental data in the form of the table so this component provides a data structure to do this.
Composition	Table structure module
Interface specification	This component just includes one module and its interfaces are the also the composed module's interfaces. Please refer to the module interface specification found in the appendix D

7 output component

Name	show outputcomponent
Role in the system	Application component
Service	Show outputcomponent is specially designed for the output of the result data. This component is used when the experiment is done and the result data is ready for the use.
Composition	Show outputmodule
Interface	This component just includes one module and its interfaces are

specification	the also the composed module's interfaces. Please refer to the module interface specification found in the appendix D
---------------	---

Appendix F The procedure for adding a new constitutive equation

In this appendix, the step-by-step instructions are presented for the designer to add a new constitutive equation as a new component into the *Virlab* software. In the following steps we mention COM, Visual C++ and Visual Basic. It is the responsibility of the designer to learn how to program in Visual C++, Visual Basic and COM technologies.

1. Read the MIS carefully, especially Sections D. 3.26 and D. 3.27 (together with Appendix D) that describe the MIS for the *displacement constitutive calculation module* and *load constitutive calculation module* and understand the corresponding module interface.
2. Follow the COM standard to write your programs about the approximation of the new constitutive equation and create the Dll file. (Many programming languages support the COM standard. We used ATL COM AppWizard in Visual C++ 6.0 to write the program for the approximation of the Maxwell equation in the uniaxial displacement-controlled experiment.) Record the component name that you declare in the Dll file. We suggest that the best approach is to have the component name that you declare in Dll file and the name of Dll file the same.
3. Register the Dll into the operating system. (In Visual C++, the Dll file performs registration after successfully linking.)

The above steps are about creating the Dll library. Below we describe how to modify visual basic source codes for the *Virlab* software.

4. Open userinterface.vbp project in the Visual Basic environment (suggest Visual Basic version 6.0)

5. Modify the constitutive equation specification module

Find the frmSetup frame and view its source codes, locate ShowConstitutiveEquation procedure and search the comments “Modify here if adding a new constitutive”. There are two tips in this procedure, “add a new item to the list constitutive equation”, add the name of the Dll file to the list; “declare a new text for the material property” declare a new Textbox for inputting each material property of the new constitutive equation.

6. Modify the Experiment module

Find the frmNext frame and view its source codes, locate DoingExperiment function and search the comments “Modify here if adding a new constitutive equation”. There are several tips in this procedure, such as: “declare a new component”, declare a new component in Visual Basic by setting a name as a new component name (For example, Set DispConstitutive = New ConEquDisp, the DispConstitutive name represents a name given by the designer now and ConEquDisp is the name you have declared for the component in the Dll file.)

7. Add a condition template “If ----End If”

If the output of your program is for the strain, find the comments “strain information” in DoingExperiment function, if the output of your program is for the stress, find the comments “stress information”. After locating the strain and stress information, add a condition template, the condition is that the current item in the list of constitutive equation (lstConstitutive) is equal to the component name you have declared in your Dll file.

8. Fill into the table

Once you locate the position of the strain and stress, you can fill the data into the table between if and end-if. First convert your output numerical value to string type, then add the output into the table by calling the table interface.

When the described eight steps have been completed, click the button "Run" in the Visual Basic environment and run the Virlab. The new constitutive equation will be available for future users of the *Virlab* system.