MODELING SOFTWARE
for learning and doing physics

DAVID HESTENES
Arizona State University

Abstract. *This is the initial report of a long term research program on the design of integrated computer software systems for physics education specifically, and math-science education generally. The program is grounded in a theory of instruction which is centrally concerned with the construction, validation and use of scientific models for objects and processes in the real world. It aims to develop a detailed theory of software design which supports and coordinates all aspects of conceptual and computational modeling. The theory is applied to the design of a versatile* Modeling Workstation *for learning and doing physics by computer, including the coordination of both experimental and theoretical activities.*

The personal computer revolution confronts physics education with an unprecedented challenge and opportunity. Microcomputer hardware and software for data collection and management, for numerical calculation, symbolic manipulation and computer simulation are developing at a breathless pace. Processor power, network speed and capacity are no longer barriers to real-time processing of complex data sets and programs in the educational domain. For most practical purposes, all hardware constraints on software design for personal computers will be eliminated within the next few years. Thus, the value of computers for physics research and education will be determined by the available software.

The computer has already become an indispensable tool of physics research, and soon most physicists will have personal computers with the power of a scientific workstation. This confronts physics education with the *challenge* of integrating the computer into the curriculum. The challenge can be effectively met only by creating educational software which articulates smoothly with scientific software used by researchers. Most efforts in this direction so far have achieved only meager success, owing, in large part perhaps, to the lack of a coherent theory for software design and use. This document reports on an long-term *research program,* in collaboration with Ronald Thornton and others, to develop, apply and evaluate such a theory.

The software development challenge presents physics education with an *unprecedented opportunity* to improve the pedagogical design of the physics curriculum. Educational research has already established that computers in the classroom do not enhance student learning without a carefully designed plan for their use (Thornton & Sokoloff, 1990). In other words, the pedagogy is responsible for the learning. The computer can enhance the pedagogy, but not replace it.

Part I of this document reviews a comprehensive theory of physics pedagogy with an eye toward extending its application from the design of classroom instruction to the design of educational software. This provides the foundation for a general theory of math-science software design in Part II. Specific designs are presented for software to facilitate learning and doing physics. We envision a *Modeling Workstation* as a universal machine for scientific thinking.

Part III concludes with recommendations for the design of integrated software systems for science and math education at all levels.

I. Modeling Theory for Instructional Design.

Any theory of instruction must answer the questions: "What to teach?" and "How to teach it?" In answer to the first question, most science educators agree that we should teach students to "think like a scientist" and acquaint them with " the structure and content of scientific knowledge." The problem with this two-pronged answer is that it is not sufficiently specific to guide instructional design. As Einstein once asked, "What precisely is thinking?" *Modeling Theory* (Hestenes, 1987, 1992) seeks to supply the answer with the necessary precision. It adopts the view that making and using conceptual models of real world entities, in a word, that *modeling* is central to scientific thinking. Moreover, it reduces the content core of physics to a small number of basic models interrelated by theoretical principles.

The view that models and modeling are central to scientific research and applications is widely shared by scientists. Indeed, it has been officially incorporated into *Project 2061*, an ambitious program of science education reform supported by the *American Association for the Advancement of Science*. However, a close look at the concepts of *model* and *modeling* reveals that they are slippery and multifaceted. They must be thoroughly explicated and analyzed before they can play a vital role in pedagogical design. That is the main task in the next few sections.

Modeling theory has recently been implemented in a high school physics curriculum and classroom tested with noteworthy results (Wells *et al.*, 1994). Details need not be discussed here, because implementation in software is necessarily quite different, but some common conclusions from educational research apply to both: First, emphasis should be placed on student *learning* rather than any act of teaching. The primary function of the teacher is to create an *environment* which enhances student learning. Second, design of this environment depends critically on the available modeling tools. Computers have the potential for enormous advances on both counts: Computer environments can be exceptionally engaging and versatile. Moreover, as shown in Part II below, the computer enables us to create a host of new modeling tools with unknown potential to enhance student learning and scientific thinking. Educational research on how to use such tools effectively in instruction must await their software implementation. For that reason we set aside, for the time being, the question of "How?" to concentrate on "What? to teach."

1.1 Why modeling?

Our emphasis on models and modeling in the design of physics instruction and software is motivated by two general claims:

(1.) By engaging students in modeling activities, their classroom experience is aligned with the practice of working physicists.

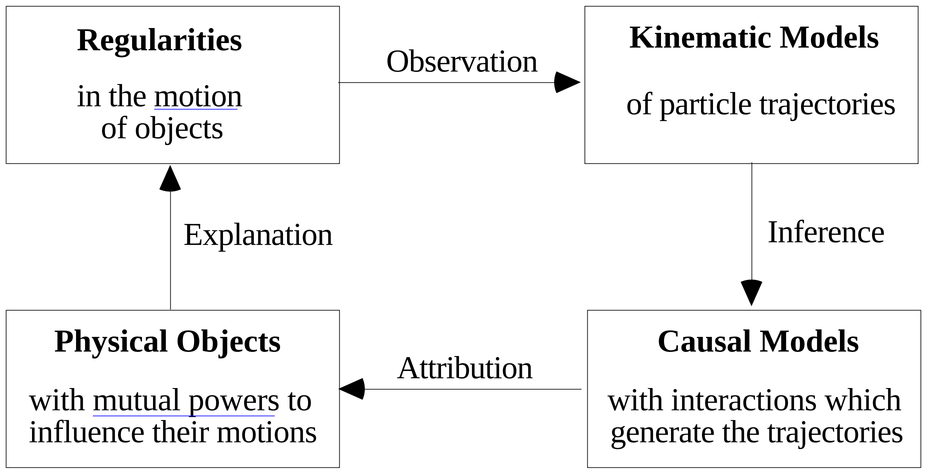
(2.) By focusing on basic models, the structure and coherence of scientific knowledge is made more evident to students by making it more explicit.

Justification for these claims comes from the study of scientific practice. We cannot do better than examine the work of Newton, who was the first to engage in systematic mathematical modeling (Hestenes, 1992). He was quite explicit in describing his modeling approach, though the term "modeling" was not in his vocabulary.

Reflecting on his research method in the preface to his *Principia*, Newton remarked that boils down to this: From the motions of material bodies infer the forces, and from the forces deduce (and so predict) their motions. This describes the essential core in a cyclic process of developing and deploying mathematical models. Call it the *Newtonian Modeling Cycle.* A more detailed schema for the cycle is shown in Fig. 1.11. Let us follow Newton in a run through the cycle. In accord with Fig. 1.11, from the best available data supplied by naked-eye observations of planetary motion, Kepler extracted several regularities codified by his three laws. Together, these three laws comprise a kinematic model of planetary motion. From this model, Newton used his laws of motion to infer a gravitational force modeled by an inverse square law. Thus, he arrived at a single causal model of planetary motion. From the causal model he was able to derive all three of Kepler's laws and thus explain the regularities observed in the first place.

Fig. 1.11

Newtonian Modeling Cycle



The Newtonian modeling cycle is generalized in Fig. 1.12 to a *generic modeling cycle* characterizing the cyclic interplay between constructing models to characterize systems in the real world and empirical evaluation of the models. An explanatory model differs from a descriptive model of the same system in positing that the system posses some property that has not been (or perhaps cannot be) directly observed. Such a property nevertheless plays a role in generating observed regularities. If the property is described as an interaction between objects, we have a causal model (a particular kind of explanatory model). In the causal models of Newtonian mechanics the unobservable properties are forces.

The modeling cycle, as described in Fig. 1.12, contains an inherent epistemological presumption, namely, that *we come to know real objects (their properties and processes) only by constructing models to represent them in the mind.* Accordingly, a scientific theory is a kind of "cognitive window" through which we "view" real objects by constructing conceptual models. This epistemological stance is aptly called *constructive realism* by Ronald Giere (1988).

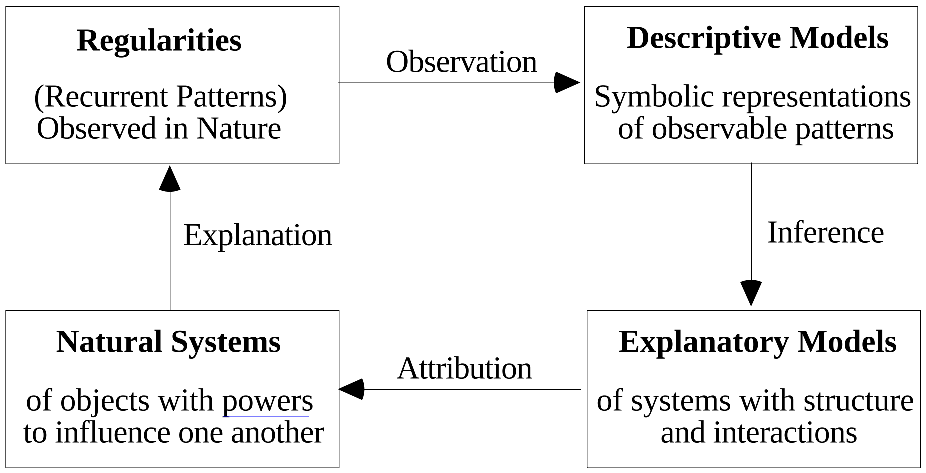
The modeling cycle (Fig. 1.12) should be compared with the *perception-action cycle* depicted in Fig. 1.13. The similarity is not accidental. The latter characterizes the interaction of a person with objects in a real world situation. Sense data is processed by the nervous system to produce a perceptual representation (an internal descriptive model, if you will) of the situation. Next, as a plan for effective action, a predictive mental model is generated. Execution of the plan with action changes the real world situation, producing new sensory data, and the cycle repeats. Of course, in every person this kind of feedback cycle is operating continuously in real time to help control behavior.

The Modeling cycle in Fig. 1.12 is a purely conceptual cycle. However, in analogy with the perception-action cycle, it has a variant in which an explanatory model is employed to *plan experiments* (actions!) on natural systems and *predict* regularities to be observed. Note that a prediction can be regarded as an explanation of facts which have not yet been observed. Therefore, predictive and explanatory models are not different in kind; their different names express only a difference in their intended use.

The modeling cycle describes a coordinated sequence of cognitive processes for *matching* conceptual models with real systems and situations. It has many variants, depending on the intended purpose of the modeler. These are discussed in Section 3 as part of a more complete account of the modeling process.

Since the modeling cycle is fundamental to the way science works, it should have a prominent place in instructional design. An instructional cycle which guides students though a

Fig. 1.12
Generic Modeling Cycle


Regularities
(Recurrent Patterns) Observed in Nature
Observation
Descriptive Models
Symbolic representations of observable patterns
Inference
Explanatory Models
of systems with structure and interactions
Attribution
Natural Systems
of objects with powers to influence one another
Explanation

classroom adapted version of the modeling cycle has been implemented and evaluated with excellent results (Wells *et al.*, 1994). After a few passes through the cycle, students have learned enough about scientific inquiry to conduct systematic investigations on their own. Of course, success of the classroom implementation depends heavily on the details about models and modeling that have been incorporated.

When the modeling cycle is built into instruction, students learn how to coordinate the complex cognitive processes of modeling to achieve definite goals (Wells *et al.*, 1994). This is an antidote for the fragmentation of procedural knowledge that results from teaching specific cognitive skills in isolation from others. In a similar way, focusing instruction on models helps students achieve a coherent understanding of science content knowledge and so facilitates its use in new situations. This assertion raises important questions about the relative roles of models and theories in instructional design.

A common view of scientific knowledge is that is hierarchically organized, and it is applied to solve problems in physics by reasoning deductively from general principles to specific empirical predictions and other conclusions. From this view theory is primary and models are peripheral if they are recognized at all. A contrary view is that scientific knowledge is organized primarily around models with theory playing a secondary role (Giere, 1988). For the purpose of instructional design, there are good reasons to adopt the second view. The first view does not give models their due. The second view admits a clean distinction between the roles of "model" and "theory" in scientific knowledge: The *content* of a science is a population of validated models; the *structure* of the science is a system of theoretical principles interrelating the models. In other words, *models comprise the content* while *theory defines the structure* of scientific knowledge. Models are primary to theory in the sense that there is no structure without content.

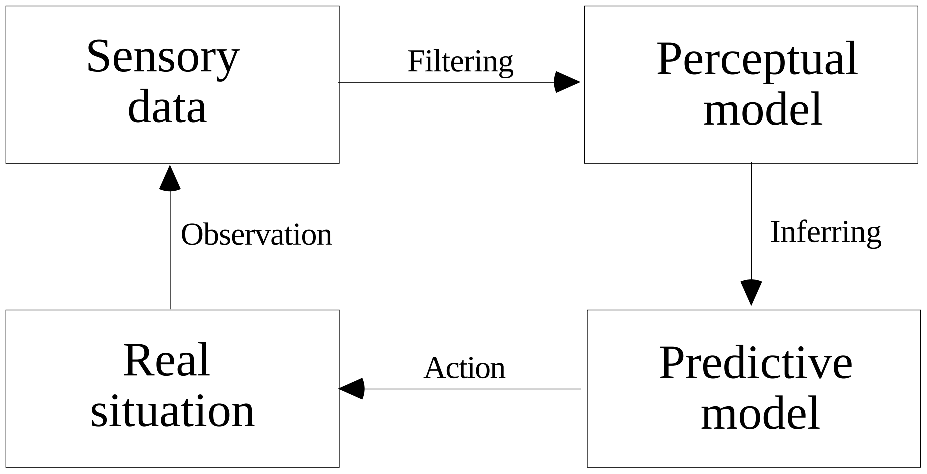
There are both epistemological and cognitive reasons for emphasizing models ahead of theory in instruction.

Epistemological reasons:

(1) *Models are specific; theory is generic.* Models are more salient to students than theoretical principles, because they refer directly to individual concrete objects in many cases.

(2) *Models are testable; theory is abstract.* Theoretical principles are empirically testable only through instantiation in models. In other words, theory relates to the real world only through models.

Fig 1.13
Perception-Action Cycle



(3) *Models are basic units of coherent scientific knowledge.* The coherence and completeness of a theory is evident only in its models. For example, in constructing particle models with Newtonian theory, all of Newton's must be applied in a coordinated (coherent) way. Aside from particular models, there is no way to assess the relation of Newton's laws to one another.

Cognitive reasons:

(1) *Abstraction*: Theoretical principles are best learned by abstraction from models, because *models supply context* for the principles. In other words, models provide a bridge between theory and reality. Failure to build such bridges contributes to serious misconceptions about physics, as in the pseudoscientific field of *energetics,* which reifies the theoretical concept of energy into a basic stuff of the universe.

(2) *Categorization*: Research in cognitive linguistics shows that people naturally classify objects by *resemblance* to mental prototypes rather than by identifying a unique set of *properties* (Lakoff, 1987). "Natural classification" is therefore fundamentally different from "scientific classification." Models can bridge both schemes by serving as *prototypes* for natural classification by pattern matching and as *exemplars* for scientific classification by properties.

This completes our justification for the primary focus of Modeling Instruction. In summary, a list of instructional objectives is given in Fig. 1.14.

2. What is a model?

This discussion of models summarizes and extends that of Hestenes (1987, 1992), where additional details can be found. Although there is a great variety of models in science, they share a common set of characteristics which reflect common characteristics of all natural systems in the real world. Fig. 1.21 lists the general characteristics specified in most, if not all *scientific models*.

The model with its characteristics is a representation of the system and its characteristics, including its parts and their properties. Let us run through the list of specifications in Fig. 1.21:

Fig. 1.14

Primary Objectives of Modeling Instruction

● To engage students in understanding the physical world by ***constructing and using scientific models*** to describe, to explain, to predict and to control physical phenomena.

● To provide students with ***basic conceptual tools*** for modeling physical objects and processes, especially mathematical, graphical and diagrammatic representations.

● To familiarize students with a small set of basic models as the ***content core*** of physics.

● To develop insight into the ***structure*** of scientific knowledge by examining how *models* fit into *theories****.***

● To show how scientific knowledge is ***validated*** by engaging students in *evaluating* scientific models through comparison with empirical data.

● To develop skill in all aspects of modeling as the ***procedural core*** of scientific knowledge.

I. The first step in specifying any model of a system is to "model" its *organization* with a *system schema* consisting of names or icons designating *constituent parts and connectivity* of the system as well as relevant connections of the system with objects in the environment (external *agents*). The system schema may be specified by an "organization chart" or with a diagram, as in Fig. 1.22.

II. The second step is to represent the basic properties of the system and its parts by "property variables" (or *descriptors*). Intrinsic properties are represented by descriptors of two types: object variables and state variables. *Object variables* are parameters of the model with fixed values (such as mass, charge and inertia tensor in mechanics). *State variables* (such as position and velocity in particle mechanics) represent intrinsic properties which may change with time. *Interaction variables* represent properties of the connections. In mechanics models they are typically forces or potentials. In other models they may represent flows of material or even information.

III. System structure is specified by *interaction laws* which relate interaction variables to state variables. Examples are Newton's law of gravitation and constitutive equations such as the perfect gas law. Note that this concept of system structure is distinct from and dependent on the system's organization specified in **I**. The organization specifies which objects are interacting (connections), while the structure specifies how they interact. It follows that the system has an *external structure* characterizing interactions with agents in the environment.

IV. System behavior (temporal structure) is specified by laws of change which relate *change* in state variables to interaction variables. Examples are Newton's second law in particle mechanics and Euler's law for the rotational dynamics of a rigid body.

Model specification is usually distributed over several representational modes: words equations, diagrams, etc.. Various graphic representations are shown in Figs. 1.22 and 1.23, with a rough classification according to their roles in model specification. The *schema* in Fig. 1.22A describes the organization of a system with three component parts interacting with three agents in

Fig 1.21
Model specification

Object / system	**Model**
I. Organization	**System Schema**
● composition	● (internal) constituents
● environment	● (external) agents
● connectivity	● connections
II. Basic Properties	**Descriptors**
● intrinsic	● object variables e. g. m, q, I
	● state variables e. g. $\mathbf{x}$, $\mathbf{v}$
● interactive	● interaction variables e. g. $\mathbf{F}$, V
III. Structure (internal/external)	**Laws of interaction** e. g. $F = GmM/r^2$, $PV = nRT$
IV. Behavior (temporal structure)	**Laws of change** e. g. $m\dot{\mathbf{v}} = \mathbf{F}$, $\dot{\mathbf{L}} = \mathbf{T}$

the environment. A *map*, as in Fig. 1.22C, describes geometric properties of the system. Note that Fig. 1.22A is a schema for the system of three blocks in Fig. 1.22C interacting with ramp R, string S and earth E. The interaction map in Fig. 1.22D describes the *structure* of a two particle model of the system in Fig. 1.22C.

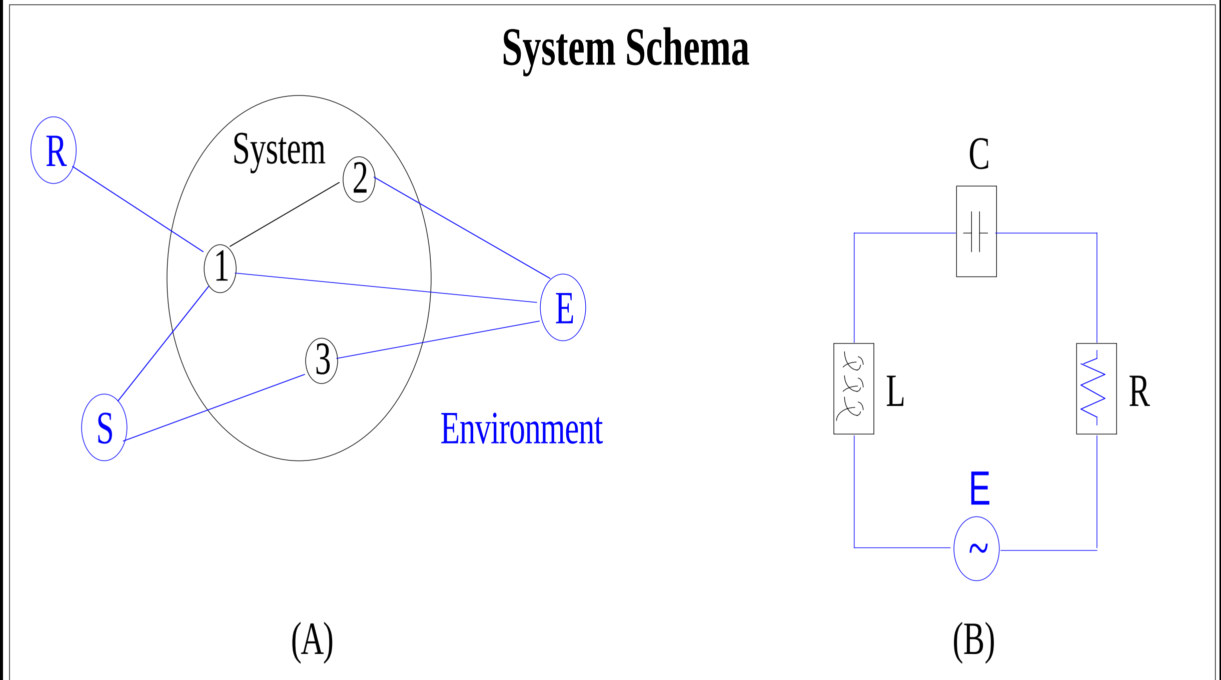
Schema diagrams like Fig. 1.22A are never used in mechanics, but they appear with various guises in other places. Note, for example, that the "electrical circuit" diagram in Fig. 1.22B is a schema for a four-component system with special icons to indicate a model type for each part. It may facilitate student learning and understanding to introduce a unified system for representing system schemas in all parts of physics.

Likewise, it may be beneficial to classify and standardize the various modes for representing system behavior. Fig. 1.23, for example, illustrates two different modes for representing change of state.

It is often convenient to *conflate the symbolic representation with the model* for a system, but it is essential to separate them sometimes to avoid profound mistakes. It must be recognized that a model meeting the specifications of Fig. 1.21 is typically distributed across several symbolic representations, as suggested in Fig. 1.24. In other words, complete understanding of a model requires coordination of multiple representations. Each representation has its own strengths and limitations. Often, two representations represent the same information in different forms (as a graph or a spread sheet, for example). At the same time, each may contain information that the other does not.

It must also be remembered that the various symbolic representations in Fig. 1.24 have significance only when there is a brain to interpret them – a brain in which they are interpreted in a *mental model* of the system. Note that the *mental model* is the result of the brain interacting with the symbolic representations, so there would be no model without both the symbols and the brain.

Fig. 1.22
Representations of System Organization and Structure



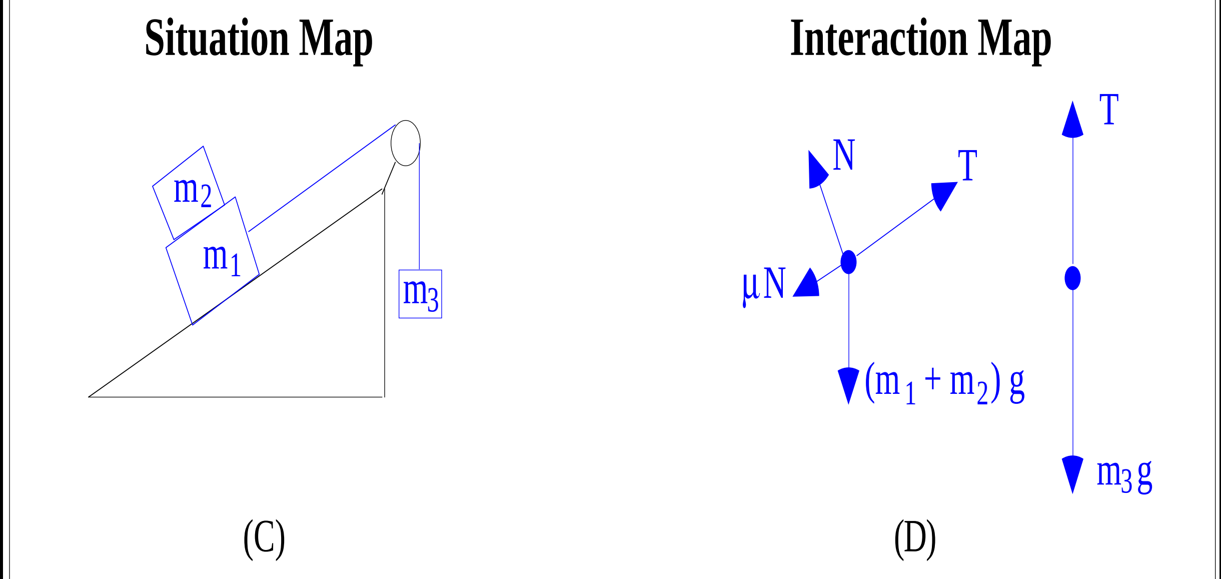
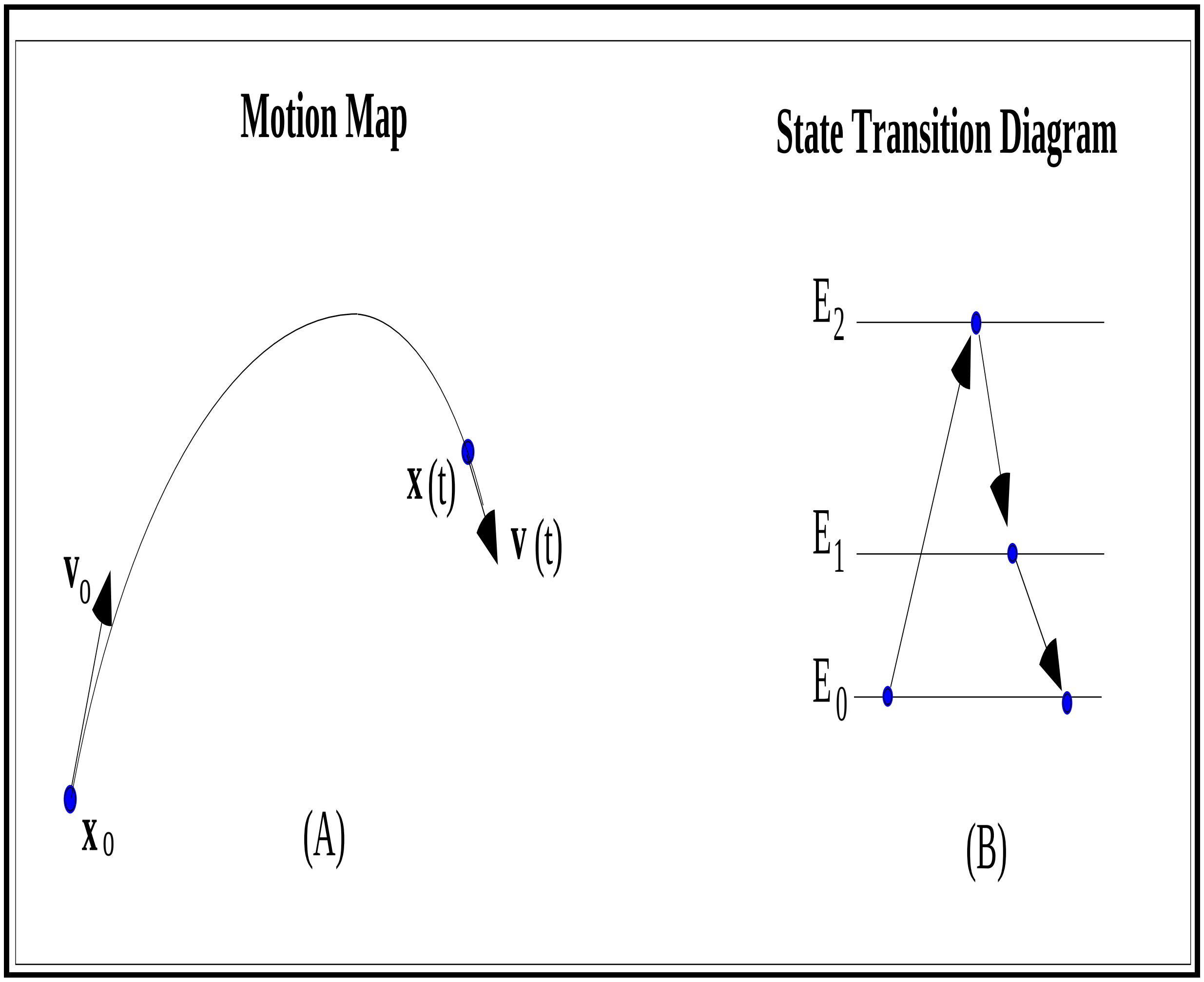


Fig 1.23
Representations of System Behavior (Process)


Motion Map
State Transition Diagram
E_2
E_1
E_0
$\mathbf{x}(t)$
$\mathbf{v}(t)$
$\mathbf{v}_0$
$\mathbf{x}_0$
(A)
(B)

Symbolic Representations

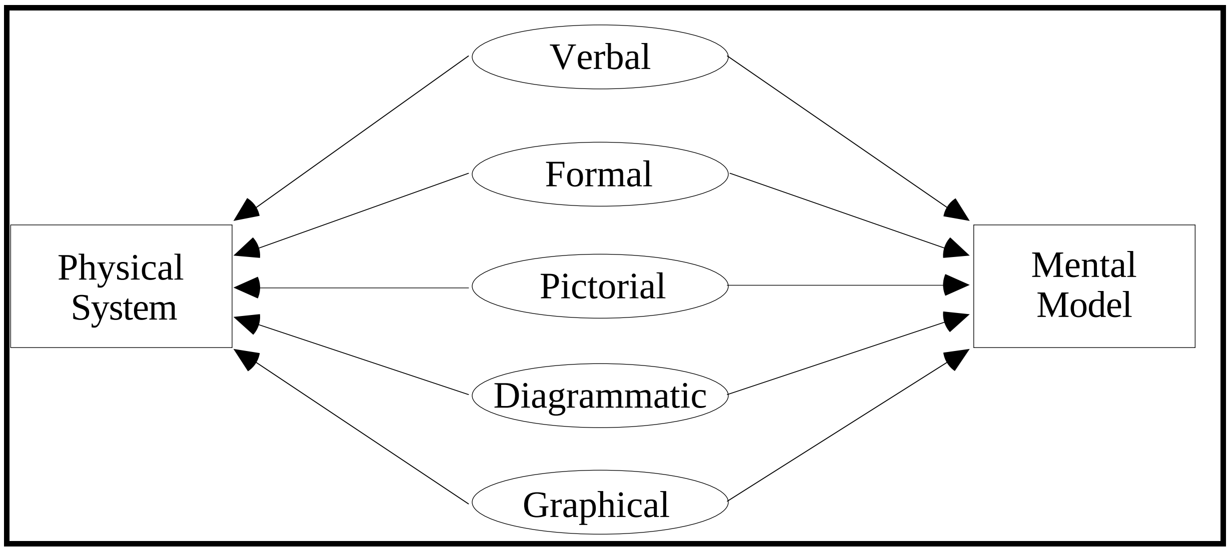


Fig. 1.24. **Multiple representations of a physical system and its model.**

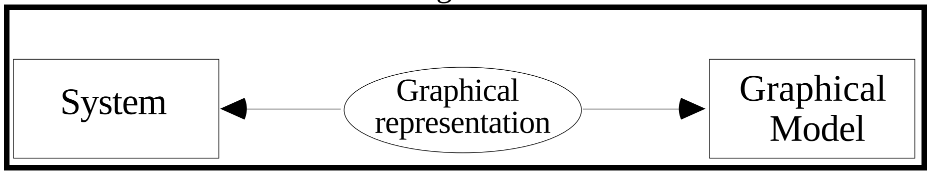
The symbolic representations induce different mental models in everybody, student and scientist alike (Redish, 1994), so how is it possible to speak of a unique objective scientific model? To answer that important question, note that structures on the left side of Fig. 1.24 – the system, the symbolic representations and the ways they are related – are *objective*, in the sense that they are open to inspection by anyone. Structures on the right side of the figure – mental model and its relations to the symbols – is *subjective*, so not open to inspection by others. However, they can be inferred to some extent from the way the symbols are used. Different individuals can *negotiate the meaning* of symbols by considering different ways to use the symbols until agreement is reached on how to use them. Though each scientist has his/her own mental model of a given system, they can nevertheless reach a *common understanding* of how to use symbolic representations for it. This common understanding characterizes a *conceptual model* of the system, accessible, in principle, to everyone and independent of idiosyncrasies in the mental models of individuals. Scientific models meeting the specifications of Fig. 1.21 are to be understood as *conceptual models* in this sense.

With the understanding that all scientific models are "conceptual," we can safely omit that adjective when speaking about them. A host of other adjectives are often used to characterize or differentiate model types, so a brief catalog may be helpful to clarity the distinctions they express.

(1) *Property type*. Models differ widely in the properties they attribute to real systems. Natural systems may be classified as *physical, chemical, biological, economic, etc*., and their corresponding models are given the same classification. Actually, the same system may be modeled as one type or another, depending on the objectives of the modeler. Any system, for example, can be modeled as a physical system, for every system has physical properties while most do not have biological properties. Each scientific theory is distinguished by a characteristic set of properties it models. Mechanics, for example, models only those properties of material objects which are relevant to motion.

(2) *Structural type*. Models differ in the structure they attribute to systems. According to Fig. 1.21, they can differ in object structure or behavioral (temporal) structure. Models which emphasize the latter over the former are called "process models." Typically, a *process model* is aimed at modeling change in a single property variable; for example, the storage, flow and

Fig. 1.25



dissipation of energy. Process models are often *partial* (or incomplete) *models*, because they ignore relevant object structure of the underlying systems. This sometimes leads to paradoxes or mistakes, as in the case of energetics mentioned earlier.

With respect to *object structure,* models can be simple or complex. A simple model attributes *no internal structure* to the object; as, for example, a particle model in mechanics or a "black box" model in engineering. Such models are often said to be "structureless," but, in the sense defined earlier, a particle has *external structure* if it interacts with something else. Only a "free particle" lacks both internal and external structure. If a model has no external structure, then it (or the system it represents) is said to be *closed.* Otherwise, it is said to be *open*.

In Newtonian mechanics, every material system can be modeled as a system of particles. In physics generally, all models can be classified structurally as *particles or fields* or some combination of the two.

(3) *Intended use*. Models may be described, according to their intended use or purpose, as *descriptive, explanatory, predictive, prescriptive, etc.*. The import of the first three adjectives has been discussed already. A *prescriptive model* is intended as a *plan for constructing* a system (such as instruments or apparatus for experiments) *or for controlling* a system (as in the design of an experiment). The variations are legion. Prescriptive models are a prime concern of engineering.

(4) *Representational mode.* Models are sometimes described by the representations used to construct them. One may speak of a *graphical model*, for example, which has been constructed by curve fitting to empirical data. Again, it is worth emphasizing that the actual *graph* (or graphical representation) must not be confused with the graphical model needed to interpret it (Fig. 1.25), though it is usually convenient to suppress the distinction. Also, note that graphs usually appear in concert with other representations, such as an algebraic equation for the graph, so the term "graphical model" should be used loosely.

The term "mathematical model" is widely used and abused in science, especially in physics, so clarification of its meaning is important. Strictly speaking, it means that the *structure and behavior* of the model is characterized by mathematical equations, so that some or all of the *property variables have quantitative representations*. It must be recognized that other representations are needed for complete specification of the model, in particular, to specify its interpretation. On the other hand, for the purpose of analyzing structure, it is convenient to separate mathematical models from their interpretation and regard them as *abstract models.* Indeed, abstraction is often taken for granted when speaking of mathematical models. This kind of *abstraction* is an enormously powerful modeling device, for the same *abstract model* can be used to characterize the structure of a great variety of systems simply by assigning different interpretations to the descriptors. For example, the *harmonic oscillator* is an abstract model which can be variously interpreted to model the motion of a mass on a spring or the flow of charge in an electrical circuit.

(5) *Embedding medium.* To have concrete existence, a model must be embedded in some concrete medium (or system). Graphical and algebraic representations by marks on paper are only partial embeddings, because a person is needed to interpret them. A complete embedding of a conceptual model in a concrete medium is called a *realization* of the model.

It has already been noted that conceptual models have realizations as mental models, but that is not the only way they can be realized. When a concrete system is built and operates according to a *prescriptive model*, it is a realization of the model. Strictly speaking, any concrete system that is accurately represented by a conceptual model is a realization of the model.

However, the term "realization" is not often applied to a system unless it was intentionally constructed to meet the specifications of its model.

Two physical systems characterized by a common abstract model have analogous structure/behavior and are said to be *analog models* of one another. This has practical significance, as when an electrical circuit is constructed to model a mechanical system or vice-versa. Analog models representing real time behavior, as in this case, are called *simulations* or *working models.* Similarly, *scale models* are often constructed to represent geometrical structure, especially of large or small systems.

A *computer model* is the realization of a conceptual model in the computational medium of a computer. Because computer models are so easy to construct and manipulate in any representational mode, and computers have unsurpassed powers for computation, visualization and simulation, realization in computers is sure to become the dominant mode for modeling in science, if, indeed, it is not so already. The same can be confidently predicted for science education.

1.3 What is modeling?

Modeling is fundamentally a cognitive process, because its primary purpose is the making and using of *conceptual* models. It is not purely a mental process, however. The perception-action cycle (Fig. 1.13) is involved: in constructing external representations on the one hand, and in relating them to real systems on the other.

Modeling has many variations with different purposes. These variations could be (and usually are) regarded as independent activities, but that disguises their common primary purpose and their conceptual elements. To emphasize their underlying unity, the various modeling activities are characterized below as variations of a single modeling process, differing mainly in their emphasis on one or another of the component processes. This unified approach is made possible by the unified concept of model presented in the previous section.

The modeling process is schematized in Fig. 1.31. Let us run through a general description of its components before considering its variations. Modeling often begins with a real world *situation*, which may be presented directly in experience or indirectly in the form of data and/or a verbal report, etc.. The first step in modeling is to *identify the system* to be modeled, including its relevant properties (identification of variables). From this a *system schema* is constructed and property *descriptors* are selected. Then a complete *model* (meeting the specifications of Fig. 1.21) can be *constructed* or *selected and adapted* from a collection of available models.

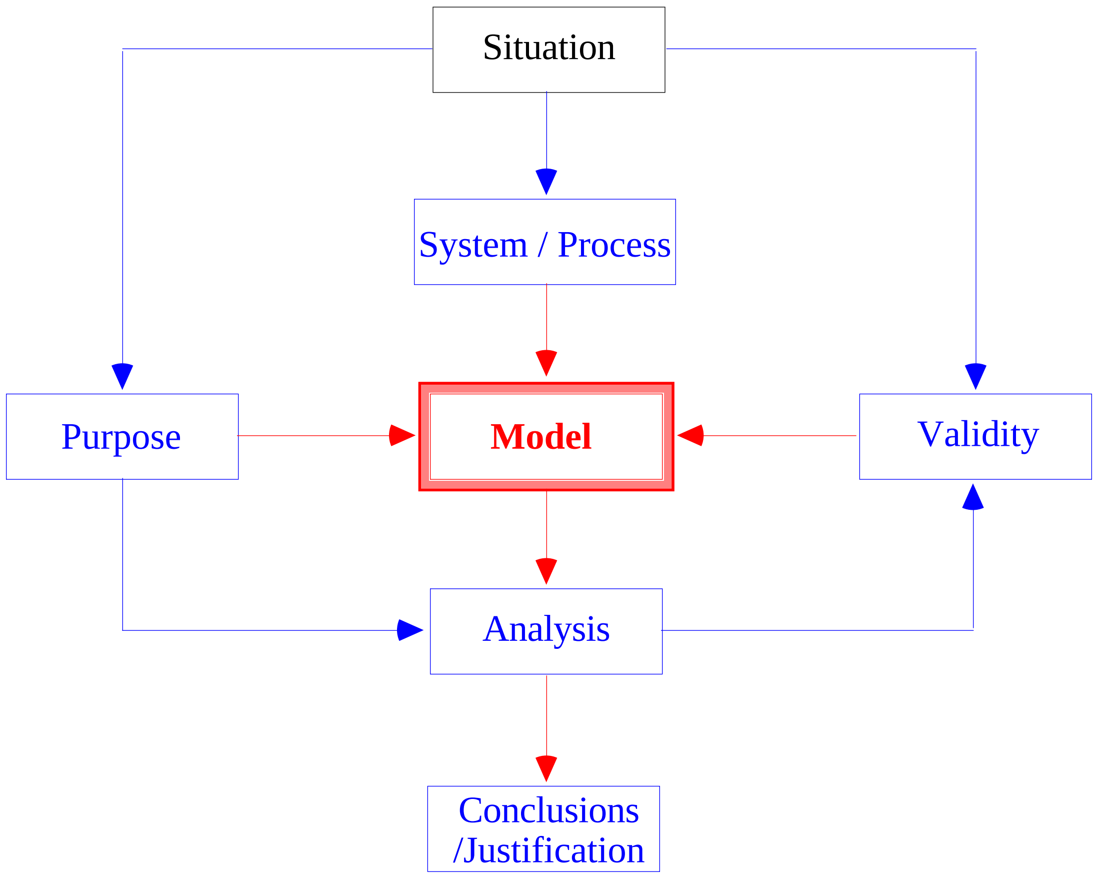
Model construction or selection is greatly influenced by the intended use or *purpose* of the model, and there is a variety of purposes that govern variations of the modeling process. The assumed purpose, for example, governs the choice of variables and the level of detail in the model. Sometimes only a crude model in desired. For other purposes, a whole family of models may be appropriate to characterize the system in successive levels of detail. Often the purpose is to answer some question about the system.

Empirical determination of the model's *validity* is achieved by comparing model with the system in the original situation. Model validation can be as complex as designing and performing an experiment or as simple as checking the answer to a problem. Validation of the model provides *justification* for conclusions about the system and situation which are drawn from the model. The extraction of conclusions from the model requires *analysis* of the model's structure and behavior as well as considerations of validity.

The modeling process as just describe can be decomposed into four major components or *modes*: model *construction, analysis, validation*, and *deployment* (Hestenes, 1987). Each mode can be a complex process in itself and is often pursued independently, with only loose connection to the other modes. Ultimately, however, science requires coordination of all the modes.

Fig. 1.31

The Modeling Process



A major objective of physics is to develop a repertoire of *validated models* which can, in principle, be deployed to account for any physical situation. *Model development*, therefore, requires the coordination of model construction, analysis and validation activities. Development of validated basic models, such as models for the electron, viscous fluids, semiconductors and lasers, is a long and complex undertaking involving both theoreticians and experimentalists. All of the modeling modes, however can be activated at various levels of complexity, some of which are suitable for students. Let us consider each one briefly in turn.

(1) *Model construction.* Models may be constructed to meet either empirical or theoretical specifications. Two primary purposes for model construction are (a) to account for observed phenomena, and (b) to elucidate and evaluate implications of a theory. In the first case, modeling begins with an empirical situation, as already described in connection with Fig. 1.31. The second case may be regarded as beginning with a theoretical situation (a thought experiment, for example), and it often ends with an experimental test of the theory.

There are no hard-and-fast rules for model construction in every case, but theories provide strict guidelines in their scientific domains (Hestenes, 1987,1994). Moreover, the model specifications in Fig. 1.21 provide criteria for full specification of a model.

(2) *Model analysis* is concerned mainly with extracting information from a model. For mathematical models this is often characterized as deductive inference. Inference from models distributed across multiple representations is called *heterogeneous reasoning* by Barwise and Etchemendy (1991), who advocate generalizing the concept of *deduction as information extraction* to cover this case.

Model analysis for its own sake investigates *ramifications* of the model with no particular application in mind. One objective is to discover conservation laws and other invariants of the model. Another objective is to "understand" chaos in nonlinear models. The analysis of simple nonlinear models is sufficiently complex to be a full time professional occupation.

When the model has been created for a definite purpose, analysis is directed at determining how or how well the model can fulfill that purpose – by extracting a prediction from the model, for example. Conclusions may be affected by information about the validity of the model, so that should be considered in the analysis.

(3) *Model validation* is the process of assessing the *validity* of a model, that is the adequacy of the model to characterize a real system. The relation of the model to the system is one of *similarity*, specifically similarity *with respect to structure and behavior* (Fig. 1.21). Therefore, a model's validity is an assessment of similarity. This assessment has both qualitative and quantitative aspects. Assessment of the model's system schema and choice of descriptors is qualitative. Comparisons of functional relations among the descriptors (laws) with empirical data can be made quantitative. The result is sometimes called "goodness of fit" or *accuracy* of the model.

Full assessment of a model's validity includes an evaluation of its *fidelity.* That involves an account of *relevant features* of the system and its environment which are *neglected* in the model followed by an estimate of their effects on the accuracy of the model. Quantitative estimates of these effects are often made by extending the model to include these features. Thus, a family of models with varying degrees of fidelity may be constructed for a single system. The fidelity required of a model depends on its purpose. For some purposes a simple (low fidelity) model is adequate and therefore preferable to avoid unnecessary complexities. For high accuracy a high fidelity model may be needed.

Models are often intended to represent (or define) a whole class of (real or possible) systems with different members distinguished by different values for the parameters of the model (mass, for example). In other words, the model is a *model type*. (Although, for cognitive reasons, it convenient to think of the whole class of models as represented by a single exemplar.) In this case, model validation includes determining a *domain of validity* for the model, that is, determining the range of parameters and environmental conditions for which a valid model of an individual system is obtained.

When a well-validated *model type* is applied to model a specific system, validation is reduced to determining whether it lies within the domain of validity. A direct empirical test of validity is not necessary, though it may be advisable as a *reality check!* – especially when there is some doubt about the domain of validity or the similarity match of the model to the system.

It is important to recognize that the relation of models to the systems they are supposed to represent is not an all or none affair, so it cannot be represented by the dichotomous logical relation of truth or falsity. Rather, the relation is one of similarity in some respects and to certain degrees. Validation of the similarity relation is a kind of pattern matching, and the fit can never be better than the limits of experimental error. Validity is sometimes described as "approximate truth." But "truth" is the wrong concept here. "Verisimilitude" is more like it.

One version of the validation process is called *hypothesis testing.* From the viewpoint of modeling theory, a *hypothesis* is a claim of similarity between models and real systems. (Giere, 1988). It must be understood that such a claim always entails qualifications, namely, that the similarity holds in some respect and to some degree. Unlike a model, a hypothesis is a linguistic entity, so it can be true or false. When a hypothesis has been verified, it is said to be a *fact*. When the type of similarity is a mathematical relation among state variables, it is called a *law*. Note that the law is not the fact, that verification of the hypothesis does not turn it into a law, as is often asserted. Rather, the fact is that the structural property of the model described by the law is similar to a structural property of the system to some degree. Like the model, the law can be said to be *valid*, but not true. Like models, each law has a limited domain of validity, which is sometimes limited to that of the model which "carries" it. The same can be said of theories, since they can be expressed as systems of laws. Theories are validated only by validating their models.

(4) *Model deployment* is the use of a given model to achieve some goal. Therefore various modes of model deployment can be classified according to purpose. These include description, explanation, prediction, planning design and control. The crucial role of models in all these activities is often overlooked. The point is best made in the case of "explanation," for explanation is the hallmark of scientific understanding.

The claim here is that every scientific explanation is based on a model, though often implicitly, and, the more detailed the explanation the more explicit the model. For example, a *causal explanation* of an event is often made by simply identifying a relevant *mechanism* and an *agent,* which are crucial ingredients of a causal model. For a complete explanation of the event the *conditions* for its occurrence must be *isolated,* and this amounts to specifying a complete causal model for the situation.

Causal models provide causal explanations for laws as well as for facts or events. A descriptive law represents empirical patterns which are observed under certain conditions. Kepler's laws are prime examples. As already noted, Newton explained these laws by deriving them from a causal model. The causal model itself can be derived (or constructed!) from Newton's laws plus subsidiary conditions, so it can be said the Newton's laws explain Kepler's laws by deduction. However, all explanations from Newton's laws proceed via a model, as in this case.

Explanations can be classified into two types: causal and structural (or reductive). As already noted, causal explanations require identification of a mechanism and an agent as a first step in constructing a causal model for the situation in question. *Structural explanations* explain "emergent" properties of an object in terms of its composition and structure. A prime example is explanation of the periodic table in terms of atomic structure.

(5) *Problem solving* in physics can be construed as a modeling process, but it does not involve anything new about modeling. It is mentioned here to point out that it can be reduced to one or another of the modeling modes. In fact, training in problem solving can be made systematic by classifying problems according to the models and modeling modes involved.

Students learn about modeling in physics courses primarily from solving textbook problems. Such artificial exercises often lead students to a distorted view about what physicists do. By making the modeling in problem solving explicit, the similarity to scientific practice is more obvious. Moreover, the structure of problem solving is clarified. The role of models can be made explicit by teaching students that the *solution* to a problem is a model, not just a number or an equation. The model contains the *answer* to any question posed in the problem. The Modeling schema in Fig. 1.31 provides a guide to systematic problem solving. In the typical textbook problem the situation is presented with a verbal description and the purpose is predetermined by a question. The solution can be obtained by following the modeling process as already described.

II. Modeling Software Design

The modeling theory described in Part I is now applied to the development of a comprehensive theory of math-software design. Implementation of the design is underway in collaboration with Professor Ronald Thornton of Tufts University. The complex technical problems of programming the design are not addressed here.

2.1 Traditions of Math-Science Instructional Software Development

We identify three traditions of math-science instructional software development distinguished by the professional specialties of their developers. The first is *Computer-Assisted Instruction* (CAI), and most of its developers are experienced teachers or educational researchers. The second tradition, known as *Intelligent Tutoring Systems* (ITS), has been developed by researchers in cognitive psychology, artificial intelligence and computer science. The third category consists of scientific software adapted to instructional purposes by scientists and mathematicians, so we might call it *Adapted Scientific Software.* We discuss each category in turn and relate them to our vision of math-science software design. A valuable set of design principles for educational computing derived from the CAI and ITS traditions is collected and illustrated in (Larkin and Chabay, 1992). We accept those insights and focus on weaknesses in the traditions, which we hope to rectify.

Most CAI programs are designed to teach specific topics in science or mathematics at the primary or secondary grade levels. Though we are developing more advanced software, we need to be cognizant of pedagogical insights derived from testing software at introductory levels. Most design principles for good instructional software apply at all levels of sophistication, but they are best investigated at the lower levels where pedagogy is most critical.

Over the last two decades a vast literature on CAI has accrued, but much of it is of poor quality or severely hampered by hardware and software limitations of no interest in principle. Most CAI programs, especially in the earlier days, have an algorithmic structure intended to promote skill acquisition by drill and practice. We believe that this emphasis is misplaced. Algorithmic structure derives from a more fundamental structure, the structure of symbolic tools essential to the subject in question. The design of a tool depends on how it is to be used; hence, software to promote skill development should be secondary to tool design. Accordingly, our first general principle of software design is:

I. Design tool software first!

Precisely what constitutes a good tool is for experts in the subject area to decide. However, subject expertise alone is not sufficient background for tool software design. The computer opens up new possibilities for symbolic tools that experts have only recently begun to appreciate (Kaput, 1992). This is a critical area for research in software design. Recognition of that fact has been growing among educational software developers in the last few years, and a number of tool-centered CAI software packages have been published recently. As exemplars, we mention the *Geometer's Sketchpad* (Jackiw, 1992). and the *Geometric Supersupposer* (Schwartz & Yerushalmy, 1993). However, the excellent geometry tools in these packages need to be integrated into a more complete *Geometry Tool Kit* in order to be useful in physics. We describe such a kit in Sec. 2.4.

The power of science derives from its tools, both instrumental and symbolic, which have been forged by scientists and mathematicians. To transfer that power to students, therefore, the first task of science education is to make the tools available. The students need *authentic tools*, of the same basic design as tools which scientists use. Unfortunately, the tools imbedded in most science curricula and software are poorly defined. Students (not to mention teachers) cannot distinguish the tools from the objects of study, or worse, the tools they are offered are mere toys, so the science is trivialized. The deliberate design of authentic tool software is the first step in the design of good educational software.

Authentic tools are essential to science education, but they are not sufficient. Science is organized into theories which coordinate the use of tools and the accumulation of factual knowledge. Accordingly, as our second general principle of software design, we assert:

II. Authentic scientific tools should be integrated into systems software which incorporates the structure of scientific knowledge.

The design of such systems software is a central problem of our research program.

The shift from algorithmic to tool-centered software is in accord with a profound change in philosophy sweeping the educational research community in recent years. According to the emerging *constructionist philosophy*, knowledge is not *transmitted* from teacher to student; rather, it must be *reconstructed* by each student individually. The very term "instruction" is therefore something of a misnomer, because it suggests that something is done "to the student" rather than "by the student." The primary job of the teacher is not to transmit information but to create and manage an environment where students learn for themselves. Likewise, the right purpose of instructional software is to create a computer environment that empowers students to learn. The challenge is to design the environment so that students learn the right things as they interact with it.

Now we turn to ITS computer programs. Such programs aim to teach explicit models of procedural knowledge in well-defined task domains. Students are first engaged in reasoning tasks, then their actions are compared with a model of expert reasoning, and they are supplied with feedback to help them improve. Despite dedicated efforts by many smart people, the ITS tradition has achieved only modest success. This can be attributed to limitations in ITS designs. Our analysis of shortcomings of CAI software applies here as well. ITS models of scientific knowledge are too narrowly defined by the rule-based structure they presume. Moreover, no one knows enough yet about cognition and learning to build credible models of expert performance into tutorial software. We know a lot more about scientific tools and systems. Accordingly, as our third general principle of software design, we assert:

III. Tutorial software should subserve learning and use of tool and system software.

This is to say that tutorials should play a peripheral rather than a central role in instructional software design. Complete separation of tutorials from the tool and system software is an important feature of the global software design. The tool and system software implicitly define the *core knowledge* to be taught. The central problem of instruction is thereby reduced to teaching students to use the math-science tools with skill and insight. The design of tutorials to address this problem remains an open issue for instructional research. A variety of approaches, from "intelligent tutoring systems" to computer-assisted "cooperative learning," need to be studied. We do not know enough about student learning to make definitive recommendations.

In seeking criteria for authentic software designs, we examine the ways computers are used by scientists. Scientific computer workstations are generally of two kinds: *empirical* or *theoretical.* As an empirical workstation, the computer functions as a *universal interface* with laboratory instruments of every kind; they collect, process and display complex data from experiments and observations. As a theoretical workstation, the computer is used for complex calculations and simulations to explore the implications of mathematical models. Although empirical and theoretical uses of computers are usually separated in scientific research today, there is ample reason to coordinate them in a single scientific workstation design, since consistency between theory and empirical data is an ultimate objective of science. Thus we arrive at another general principle of authentic software design for student use:

IV. *Empirical and theoretical functions should be sharply distinguished yet coordinated.*

Given the indisputable value of computers in scientific research, physicists and engineers have recently started adapting scientific software designs to instruction. In the theoretical domain, most of the effort has gone into creating computer simulations of mathematical models or adapting mathematical software packages for student use. One of the most ambitious physics simulation packages is the heavily marketed *Interactive Physics* (1990). Its deficiencies are typical of software in this tradition. It exhibits insensitivity to cognitive and pedagogical issues and shows no trace of familiarity with CAI and ITS research. In short, it suffers from the lack of a theory of software design.

The first step in adapting scientific software to instruction is to recognize that the scientist possesses *tacit knowledge essential to operate and interpret the software*. For student use, this knowledge must be explicitly built into the software or supplied in an instructional setting. The *explication* of tacit scientific knowledge is a major task in the theoretical component of our research program. As explained in Part I, it leads to the general conclusion that the tacit knowledge is mainly about the structure of mathematical models and the processes of modeling. Thus, the tools inherent in scientific software are *modeling tools,* and the systems are *modeling systems*. *Modeling is the main activity of scientists.* It is a complex activity with many ramifications.

The greatest deficiency of simulation software like *Interactive Physics* is that it fails to explicate (i.e. make explicit) the structure of the underlying model that generates the computer animations. Beyond that, it hardly begins to help students develop coordinated modeling skills. For these reasons, *Interactive Physics* cannot be expected to improve physics instruction appreciably.

Recognizing that the main function of *scientific workstations* is to support the empirical and theoretical aspects of modeling, is to see them as *modeling workstations*. We submit that their designs are greatly improved by displaying their modeling functions explicitly on the computer screen. Futhermore, most modifications needed to make scientific software accessible to students will be valuable to scientists as well. We foresee a narrowing gap between student and scientist workstations. Accordingly, we recommend developing a progression of student workstations with increasingly sophisticated authentic modeling tools and systems leading to a full-fledged Modeling Workstation for research scientists.

2.2. The Modeling Workstation Vision

Modeling theory suggests that the *main purpose* of math-science software should be to facilitate modeling activities. Applying the general design principles of the preceeding section, we propose to develop an integrated system of math-science software modules to fulfill this purpose. We call the computer implementation of this software system a *Modeling Workstation*. The workstation should contain, in a flexible and accessible fashion, all the procedural, factual and theoretical knowledge needed for any modeling activity in physics, from data collection and analysis to mathematical modeling and simulation.

We envision a computer workstation with two complementary functions: First, to act as a conceptual *environment* which students can actively explore to discover the rich structures of physical theories. Second, to serve as a *tool*, indeed, as a *machine* for developing mathematical models and deploying them to account for real physical data and phenomena.

Beyond this, we envision a progression of Modeling Workstations with *homologous structure* suitable for student use at progressively more sophisticated levels of scientific competence. Each workstation consist of modules organized into three separable "shells":

I. Modeling tools, tool kits and stockrooms
II. Modeling Systems
III. Applications

Here we describe the general structure and function of the shells. Fuller details about the first two shells are given in subsequent sections.

The *tool shell* contains the core knowledge of modeling tools and materials. The tools are of two general types: (1) *Empirical tools* for collecting, analyzing and displaying empirical data. (2) *Theoretical tools* for formulating and analyzing conceptual models, including alphanumeric and graphical representations. The tools are designed to be authentic scientific tools; therefore, students will have a better opportunity to develop authentic scientific thinking skills as they learn to use the tools.

The tools are organized into self-contained *tool kits* that generate coherent conceptual structures. Each tool kit is accompanied by a *stockroom* of conceptual models and materials which can be constructed or otherwise manipulated with the tools. To promote software portability, tool shells are self-contained and separable from the system and application shells. Stockroom inventories can be expanded indefinitely, and tool kits are graded by complexity and sophistication. Insofar as possible, different tool kits are designed to be compatible, so that they can be combined and coordinated in complex tasks. The most sophisticated tool kits are suitable for scientific workstations, but even at that level simpler tool kits are sometimes useful.

The *system shell* organizes scientific knowledge to facilitate its use in every kind of modeling activity. The primary objectives for its design are: (1) that students learn about the structure of scientific knowledge as they explore the system shell and use it in modeling; (2) that students learn basic procedural knowledge of science from the ways in which the system shell coordinates the use of modeling tools. Although the system software contains some automated algorithms, it is mostly interactive, encouraging a dialogue with the user. The system shell contains and coordinates both generic and specific components: Modeling in each scientific domain involves *specific techniques* tailored to the subject. Modeling also involves *generic techniques*, such as systems theory, systems analysis, or matching models to data, which are applicable to all sciences. Generic modeling techniques form a nucleus of transferable cognitive skills for the entire K-16 math-science curriculum. This has implications for curriculum software design discussed at the end of this paper.

The modeling tool and systems shells comprise the *core* of the Workstation. The *applications shell* contains everything else: tutorials, simulations, games and any other imported applications consonant with the design and purpose of the Workstation. Tutorials are intended to help students learn how to apply the *core knowledge* in the tool and system shells; their use will diminish as user modeling skills improve. With the establishment of universal standards for software compatibility (now in the making), any simulation software, such as *Interactive Physics*, will be importable into the Workstation to enrich modeling activities.

As most applications will be imported rather than intrinsic to the Workstation, we can safely defer discussion of the many interesting possibilities to a later date. We mention only that, for instructional purposes, we strongly favor the construction of *model worlds* (or *microworlds*) on the computer, where the students can play with model objects and investigate their properties. Research by White (1983) and others shows that this approach has great pedagogical promise. Our Workstation provides all the tools needed to construct and experiment with model worlds, and to move from them to genuine physical applications. In this respect, it can serve as a generic pedagogical tool for the design and control of instructional environments.

The modular structure of the Modeling Workstation makes it extendable and adaptable to new modeling tasks. Although our Workstation is designed specifically for modeling in physics, by separating specific and generic aspects of modeling, we aim at a universal design applicable to all the sciences. In this way, we hope to contribute to the unification and coordination of scientific knowledge.

2.3. Cognitive Principles of Interface Design

By *Interface* we mean the human-computer interface through which a user interacts with a computer. It consists of a screen for display and some means for user control (usually a keyboard and mouse). For brevity here, we ignore audio interactions, although they may well serve a number of useful purposes in the workstation.

Interface design is critical, because it determines the quality of the human-computer interaction. Most of our remarks about software design concern the Interface. Interface design has been a rich subject for research in cognitive science for more than a decade (Norman, 1983). Kaput (1992) has produced the most profound analysis of human interaction with symbolic representations and the potential for computers to enhance it. Here we summarize some of the lessons learned about cognitive aspects of visual interface design.

Interactive Modes

The computer is an interactive medium when the user responds directly to displays on the screen and modifies them. Interaction can proceed in one of two modes: direct manipulation or command. The choice of mode is critical to good software design, so we make some general remarks about it:

(1) The *direct manipulation* mode enables the user to "grab" objects (with a mouse, stylus, space ball or data glove, etc.) and move them around the screen or perform other operations on them. It is the implementation of this mode, along with metaphors for objects and actions, such as "the screen as desktop", that makes Apple Macintosh software so user-friendly. For this reason, we introduce direct manipulation into our design of modeling tools whenever possible. The main problems with this mode are for the user to know what type of functionality the mouse click controls and what parts of the display can be directly manipulated. One solution is to prompt or cue the user with icons, menus or "balloon help". However, care must be taken not to detract from the *immediacy* and *self-evidence* of direct manipulation, which are its main advantages.

(2) In the *command mode* the user issues typed or verbal commands to the computer for actions to be carried out. It is not always possible to substitute direct manipulation for a command. For example, although an approximate value for a parameter can be set by adjusting an analog scale with a mouse, an accurate numerical value must be typed in. The main problem with the command mode is that the user may not know or recall the commands at his/her disposal, and command names are seldom derived from a coherent set of rules. A solution is to use menus which list the available commands, but this quickly becomes tedious after the user has learned the commands. The number of commands can be reduced with expandable menus which group the options. Of course, the command mode has the advantage of exploiting the user's familiarity with the English language, and it is especially useful for delegating complex or tedious tasks to the computer.

In general, good interface design exploits the best features of both direct manipulation and command modes. In particular, if direct manipulation is used for a variety of tasks it must be augmented by commands to inform and assist the user.

Models and Representations

In science, a model is a representation of a real thing (or class of things), including its structure and behavior. The representations are of two general types: descriptive or depictive. *Descriptions* can be *verbal* (using English) or *formal* (using mathematics) or some combination of the two. They consist of character strings, which are subject to syntactical rules specifying relations and operations among the characters (Kaput, 1992). *Depictions* are 2-dimensional displays such as graphs and diagrams, or pictorial analogs such as animations or maps. Many depictions (tree diagrams, for example) include labels which couple them to descriptions, so they are representations of mixed type. On the other hand, some descriptions, such as spread sheets or matrices, have depictive features.

A *mathematical model* is a formal representation consisting of mathematical equations interrelating *descriptive variables.* It is not actually a complete model, because it lacks a *semantic interpretation* relating it to its referents. The interpretation is typically supplied by a combination of verbal and depictive representations. Thus, characterization of the complete model is often distributed over representations of different type. Some features of the model have representations of different type, each with distinctive advantages. Therefore, coordination of multiple representations is an important objective of interface design.

When a model is represented (e.g. by characters and diagrams) in an *inert medium* such as pencil and paper, the *syntactic and semantic structure* must be supplied by the reader. In contrast, the computer is an *active medium* capable of representing this structure by constraints on the manipulation of symbols. This means that the user need not be fully acquainted with the structure to use a computer model; he/she learns the structure by manipulating the model, just as all of us have learned the properties of physical things by manipulating them. In other words, the structure of computer models is *learnable* by a naive user. The problem is to design *modeling tools*, for constructing and manipulating models, which make the syntactic and semantic structure as self-evident as possible.

Semantic knowledge in science is mostly tacit and indirectly acquired, so a great deal of experience is required to consolidate it: this naturally leaves the student in a confused or uncertain state. In the next paragraph we offer an important example of how to design modeling tools with *coupled* (or *hot-linked*) *representations* to make semantic relations *salient* to the student. A coupling between a formal and a depictive representation can be construed as a "model" of the

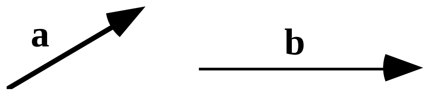
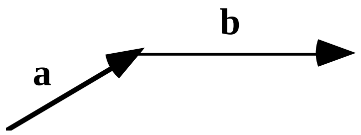
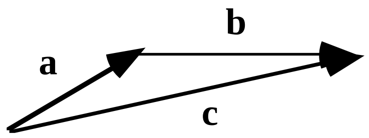
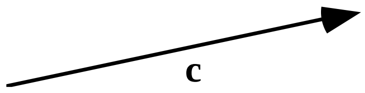
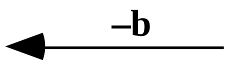
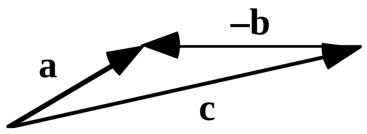
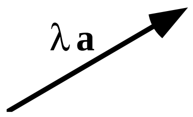
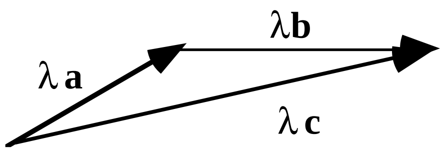
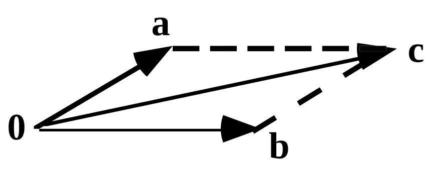
Coupled Windows		
Line	**Description**	**Depiction**
1		
2	**a + b**	
3	**a + b = c**	
4	**c**	
5	**–b**	
6	**a = c – b**	
7	λ**a**	
8	λ**(a + b) =** λ**c**	
9	**a + b = c**	

Figure 2.31. Semantics of Vector Algebra

semantic relation between a *model* (the formal representation) and its *referent* (the depictive representation). We suggest that the learning of such couplings as mental *associations* (mostly by informal means) is one of the main ways by which scientists consolidate semantic knowledge.

Vector semantics

The interpretation of *vectors* as formal representations of directed line segments (*arrows*) is essential to physics, though vectors and arrows can be regarded as different but equivalent representations of relations in the real world. Key elements in the design of a *vector algebra tool* to express the semantic relations between vectors and arrows are illustrated in Fig. 2.31 which depicts coupled description and depiction windows side-by-side on the computer screen. Actually, both descriptions and depictions could appear in the same window, but keeping them separate helps emphasize the semantic coupling.

On line 1 of the Figure, two arrows are drawn. Labeling them with characters **a** and **b** automatically *hot-links* them to the depiction window, so when **a + b** is written on the left of Line 2 the computer automatically lays the arrows tail to head on the right. When the sum **a + b** is set equal to a vector **c** on the left of Line 3, the corresponding arrow appears on the right. After that, typing that letter **c** on the left, as in Line 4, produces the labeled arrow on the right. Likewise, the minus sign on the left of line 5 reverses the sense of the arrow designated by **b** on the right. If the equation in Line 3 is solved for **a**, as in Line 6, the modified triangle immediately appears on the right. When a vector is multiplied by some scalar λ as in Line 7, a default value (near 1) is automatically assigned, and the arrow λ **a** appears on the right with an analog scale displaying the value of λ . The value of λ can be adjusted on the scale with a corresponding change in the length of λ **a**. Similar adjustments using the distributive law in Line 8 produce a family of similar triangles.

These examples suffice to show how the *formal syntax* of vector algebra can be related to a *geometric syntax* of depictions with arrows. There is a definite correspondence between rules for combining and manipulating arrows (geometric syntax) and the formal syntax governing vector manipulations (Hestenes 1986). This should not be surprising, because vector algebra was invented to provide a formal language for geometric relations.

It is important to note that the geometric depiction or interpretation of vector algebra is not unique. For example, the older literature often distinguishes between *free* and *bound* vectors. This not actually a difference in vectors but a difference in depiction (or interpretation). The depictions we have discussed so far are free vector depictions. In Fig. 2.31, the "free depiction" in Line 3 can be compared with the "bound depiction" in Line 9. In the bound depiction the tails of all vectors are "bound" to a single point which designates the zero vector. When vector algebra is used as a language for geometry, the vectors representing points (called *position vectors* in physics) are bound vectors and the point designated by the zero vector is called the *origin*. However, "parallel displacements" and other relations among points are represented by free vectors.

In applications to physics, alternative depictions support alternative physical interpretations of algebraic equations with identical form. For example, the depiction in Line 3 of Fig. 2.31 is suitable for a displacement, whereas the one in Line 9 is suitable for a superposition of two forces. The crucial role of depictions in mediating the physical interpretation of mathematical models often goes unrecognized by students (Hestenes 1987), because they are employed only informally. In practice, the physical interpretation of a model is usually specified by mapping the formal description onto a depiction and the depiction onto the physical phenomenon, for the simple reason that the depiction is easier to fit to data. These *semantic correspondences* are expressed by

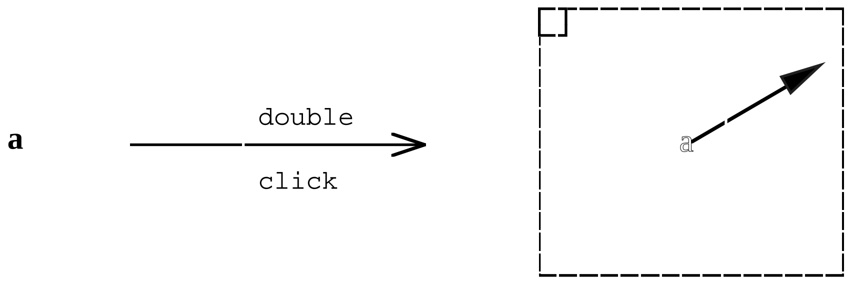
formal model <----------> depiction <----------> object

We have described how computer modeling tools can be designed to make the first semantic correspondence visible to students. Later we describe some ways to make the second correspondence visible. We submit, though, that when physicists or mathematicians speak of an interpretation of a formal model, they are usually thinking of the first correspondence. By making the semantic correspondences immediately explicit, modeling tools make them more learnable.

Abstraction

The foregoing ideas for expressing semantic relations by coupling representations of different type provides a framework for explicating the fuzzy concept of abstraction and clarifying it with computer implementations. We regard *abstraction as decontextualization*, the separation of a formal (or verbal) expression from its semantic context. Thus, an *abstract model* is a formal model separated from any interpretation. Likewise, an *abstract symbol* is a symbol detached from its referents. This is expressed clearly in the computer by deactivating the coupling between "descriptors" and "depictors", that is, between descriptions and depictions.

We can be more specific. As before, let the character **a** denote a *vector variable*. A variable is actually a function, a many-to-one function which "selects" one value from a set of "allowed values." The character **a** can be regarded as the name for a file containing the essential semantic information defining the semantic meaning of the variable. Accordingly, we design the interface so a double click on the mouse opens the file, as illustrated by

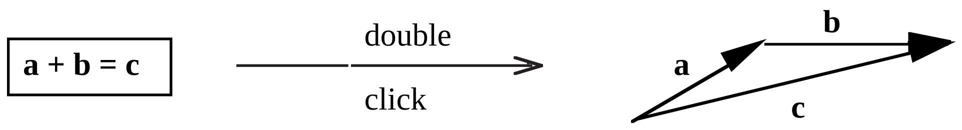


At the least, the file is a small window exhibiting an arrow with its tail attached to the place of **a** on the screen. The arrow is a *depictor* representing the value of **a**. It can be stretched/rotated about the origin by "grabbing" and moving its head with the mouse, to express a change in the value. Of course, the same device is used to indicate the value of a scalar variable on a sliding analog scale. Note how the tacit distinction between a variable and a "value of the variable" is made explicit by this device. It should be a very effective means for helping students consolidate the concept of variable, but it is not a mere tutorial device! It is obviously valuable for sophisticated users.

The "semantic file" for each variable will be modifiable. For example, the value of a vector variable can be expressed alternatively by a list of components with numerical values, but that requires introducing additional information defining a reference frame. Furthermore, the variable can be given a physical interpretation by attaching its values to a system of physical units. The semantic file thus enables us to give precise meaning to vague notions of "levels or degrees of abstraction" or "semantic distance" of a symbol from a specific referent.

There are two natural *levels of abstraction.* At the lowest level, no physical referent is specified, but descriptive characters (descriptors) do have depictive referents. At the highest level, the *formal level,* descriptors are disassociated even from "depictive meaning" to become fully abstract characters. Nevertheless, the characters retain "syntactic meaning" assigned to them by the syntax of the formal system to which they belong.

Our characterization of abstraction and semantics for symbolic characters carries over to models. For example, the equation $\mathbf{a} + \mathbf{b} = \mathbf{c}$ can be regarded as a model for a triangle if the vectors are interpreted as relations between vertices of the triangle. This interpretation can be made explicit by highlighting the equation and double clicking to call up a depiction; thus,



The depiction exhibits a specific triangle, and it can be deformed into any other triangle by grabbing and moving the vertices with a mouse. The equation is unaffected by such alterations of specific values for the variables. It is thus an "invariant property", that is, a property of all triangles. All general properties of triangles, such as the "law of sines" and the "law of cosines," can be derived from the equation using the formal syntax of geometric algebra (Hestenes, 1986). Here we have an example of how mathematics achieves a complete separation of form (or structure) from content. The computer tools we have described make all this immediate, explicit and salient. Their pedagogical potential is enormous.

To summarize, computer tools enable us to express the *interpretation* of a formal model explicitly with coupled representations. Decoupling the formal model from its interpretation lays bare the relational structure of the model. Conversely, a single formal model can be coupled to an unlimited number of different interpretations. By this means mathematics gains its great power and universality. The computer tools that implement all this explicitly thus have a great range of applicability to modeling physical phenomena.

Transfer

The *knowledge transfer problem* – how to foster the ability of students to transfer knowledge acquired in one domain to applications in another domain – is one of the most important and long-standing problems in education. We submit that modeling tools such as we have been discussing have the potential to go far to its solution. *The crux of the problem is in the tools not the teaching.*

Lessons learned from the vast but inconclusive research on knowledge transfer have been summarized by Perkins (1989). Expert thinking has been found to depend on "specific, context-bound skills and units of knowledge with little application to other domains." Transfer is highly specific and must be "cued, primed and guided; it seldom occurs spontaneously."

The possibility of transfer is highly dependent on how the knowledge and skill have been acquired. Perkins suggests that the key to transfer is acquiring general "metacognitive" skills and principles for managing knowledge. Such knowledge is first learned in the context of a specific domain and then decontextualized (or *abstracted*) for application to other domains. Without denying the importance of metacognitive knowledge, we submit that there is a more fundamental determinant of knowledge transfer. *Skills are tied to tools*, cognitive as well as physical. Without a basketball, Michael Jordan's great skill cannot be exhibited and could not have developed. Most of the cognitive tools that experts have acquired are of slap-dash design, because they have been created to meet the needs of a moment. The computer modeling tools which we have been discussing are designed to facilitate both abstraction and interpretation of mathematical symbols. The skills of decontextualizing and recontextualizing, which are essential to knowledge transfer, are incorporated into the design of the tools. One could say that these tools are designed to facilitate knowledge transfer. It remains to be seen how much use of these tools promotes transfer among students.

Attentional focus

One of the most robust observations of cognitive research is that attentional focus is crucial to learning. Students often fail to learn in an educationally rich environment, because they fail to attend to the right things. Therefore it is of the utmost importance that we design the interface display to direct the user's attention to essential features. Here we review some devices for doing that. Again, note that these devices are not mere tutorial crutches; they are helpful to sophisticated users as well.

Grouping and Chunking

The cognitive process of mentally collecting of several items to form a single entity is so basic to understanding that a number of notational devices have been invented to represent it. The parenthesis ( ) is used for this purpose in formal systems, and the dotted outline $\boxed{}$ is often used in depictive displays. For ease and simplicity, the *highlighting* feature on computers is an unmatched grouping device, and it should be systematically integrated into descriptive and depictive syntax.

The replacement of an entire set by a single character, which can function as its name, is a proven device in mathematics. Let us call it *packing*. It is closely related to the cognitive ability called *chunking*, the ability to deal with a complex mental or perceptual entity as a single unit. Indeed, we believe that *notational packing* promotes *cognitive chunking*. Standard software allows one to replace a highlighted set with a single character and even retrieve it with an "undo" command. It could easily be stored permanently with, say, a "*pack*" command, after which the character could be "unpacked" with a double click to retrieve the original set. This is the simplest kind of file creation. Our "packing device" has been built into a powerful, simple programming system called *Boxer* (diSessa & Abelson, 1986).

Suppression of detail

The computer has powerful capabilities for removing or suppressing irrelevant or distracting detail from the computer screen. Packing and abstracting devices can do that, though they have other purposes as well. A particularly important application of abstraction tools is to simulation. A (computer) *simulation* is a temporal display on the computer screen controlled by the equations of a *model*. It is thus a (partial) representation of its "underlying" model, but it should not be confused with the model itself (though it often is). We distinguish between bare and dressed simulations. A *bare simulation*, such as a moving spot on the screen, is devoid of semantic content which might suggest a referent. It becomes a *dressed simulation* by adorning it with semantic markers, such as a rocket icon riding on the moving point. In experiments with children, White (1983) has found that bare simulations have definite pedagogical advantages over dressed simulations, in keeping with our observations about the importance of abstraction.

Besides suppressing spatial detail as just discussed, the computer can suppress temporal detail in user actions by *automating* some of them. It can reduce routine or complex algorithms to a single command; moreover, software like the *Geometer's Sketchpad* (Jackiw, 1992) greatly increases the efficiency of drawing and manipulating figures with constraints. Cognitive research shows that higher-order thinking requires *automation* of lower-level operations (Anderson, 1983). By automating lower levels, the computer liberates novices to operate at higher levels. The need for rote memory is also reduced thereby, so the computer has the potential to shift the emphasis in education from memorization to the study of structure. The problem is to design computer tools which make the structure of science and mathematics as visible and obvious as possible. As Kaput (1992) observes, that is one of the main objectives of mathematical notation.

Reifying Actions

Kaput notes that students tend to focus on states rather than actions, evidently because actions are so evanescent. One of the most powerful inventions of mathematics is the *operator notation*, which reifies actions by creating notations to represent them. In other words, actions are turned into concrete objects which can be combined and manipulated on the computer screen. Students need to learn operator concepts and notations at an earlier age, and modeling tools can be designed to facilitate that by hot-linking executions, depictions, and simulations of actions to *action characters*.

Action focus

Learning requires more than attentional focus: the object of attention must be coordinated with some action, overt or covert. Such a simple device as the "forced choice" may be sufficient to consolidate the desired mental associations. For example, at some stage the user should be required to introduce labels, when setting up hot-links, to direct attention to specific associations involved. The associations may be reinforced by allowing the user to customize the notation. Later, automation of the labeling with default assignments releases the user's attention for higher-order operations.

Manipulable diagrams, hot-linked to equations which constrain them, provide an easy, even powerful, means for a qualitative analysis of consequences of the equations by *parametric variation*. Later we explain the use of *adjustable overlays* to make the matching of a model to data simple and salient.

Procedural insight

Donald Norman (1983) has studied the mental models people have of technological devices they use, such as the hand calculator, computer and video camera. They are forced by circumstances to develop such models to guide them in operating the devices. He finds the models "surprisingly meager, imprecisely specified, and full of inconsistencies, gaps and idiosyncratic quirks" even among experienced operators. The difficulties students have in understanding and applying physics is of essentially the same ilk. To operate a calculator efficiently, a person must have a clear and accurate mental model of how it works. The same is true of applying mathematical methods and will, of course, be true of operating the Modeling Workstation. Accordingly, we strive to make the Workstation as *operationally transparent* as possible, by incorporating maps of its structure and procedural plans, along with provisions for taking the knowledge system apart and reconstructing it. We exploit powerful facts about mental models: The path is a metaphor for the journey. The tool is metaphor for the task.

It should be noted that direct manipulation on the computer screen accurately reflects the structure of the underlying programs it controls.

2.4. Modeling tool kits

In this section we discuss the design of an integrated sequence of tool kits for modeling *geometry, kinematics, dynamics, and general dynamical systems*, in that order. Beginning with geometry, each kit presupposes and expands the capabilities of the kit preceding it.

Geometry tool kits

Geometry and kinematics underlie all of physics and the rest of science too, for every real thing is presumed to be located in space and time. Geometry tools are designed for *constructing, manipulating and comparing geometrical models* of real objects, especially rigid bodies. A geometrical model is one with geometrical properties alone. The geometry tools are capable of *creating and coupling both formal and depictive representations,* as described in Section 2.2. Formal apparatus for vectorial and coordinate representations will be introduced in tandem with depictive apparatus for figural representations. No sharp divisions between geometric, algebraic, vectorial and trigonometric methods are made; they are all integrated into a single system of geometric tools. Traditional synthetic methods of geometry are not incorporated into the system, because they are replaced by more powerful formal methods of modern mathematics. However, they survive, in part, in verbal descriptions of formal relations and so provide a link to existing knowledge through the English language.

Geometry tools are of three general types, which we describe briefly with emphasis on some unique features of our design.

(1) *Construction Tools*.

For 2-d Euclidean geometry, these tools are used to construct geometric objects of three kinds: figures, bodies and curves.

By *figures* we mean points, lines, line segments, circles and circular arcs, as well as any figures, such as polygons, which can be assembled from them. The tools are also able to create *constraints* among the figures, such as tangency of a line to a circle, which are preserves (invariant) under manipulations of the figures. Some excellent geometry software (*Geometer's Sketchpad* and *Geometric Supersupposer)* for doing such things is already on the market. Our tools will go beyond that in hot-linking vectorial and coordinate representations to the figures so their geometrical properties can be characterized by equations. Hestenes (1986, Chap. 2) has already developed this subject thoroughly with geometric algebra. Following his approach, we regard the concept of direction as an irreducible primitive of geometry. *Directions* are represented formally by vectors and depicted by arrows. Consequently, arrows will figure prominently in all our depictive constructions from the beginning.

Geometric figures can be turned into models of solid *bodies* by making them rigid and impenetrable. Although rigidity and impenetrability in real bodies emerge from physical forces, they are formulated as geometric constraints, that is, they are modeled as geometric properties. However, both these properties become evident only when the objects are manipulated. A convenient way to depict impenetrability is by thickening the boundary lines of figures.

More advanced construction tools can be used to create algebraic and Bezier curves, and even general analytic curves. However, most useful curves will be available from a "stockroom" (or library) of geometric models, which will be included as part of the complete geometry tool kit.

(2) *Manipulation tools.*

These tools move geometric objects about and rescale them. There is a tool for executing each of the basic transformations: *translations, rotations, reflections* and *dilations* (rescaling). These transformations are represented formally by *operators* which, when applied to any vector designating a point in an object, will automatically induce a depiction of the transformation of the entire object. Except for reflections, the transformations are controlled parametrically by the user, so they can be seen as unfolding continuously.

(3) *Measuring tools.*

These tools ascertain geometric relations. They do not alter the geometric objects to which they are applied. The most basic geometric properties are *distance* (or *length*) and *direction*. It should not be surprising that these are also basic properties of vectors, because vectors were invented to represent them. To depict measurements of these properties, we need two kinds of tool: *ruler* and *protractor.* The computer enables us to design tools which make measurements on the screen far quicker and easier than in the physical world. These tools are not just toys, though, they can be applied to make physical measurements directly off video displays. The exercise of doing such measurements helps students develop a strong grasp of "units" because of the sensitive scaling and perspective involved.

One convenient type of ruler is a kind of *expandable tape measure*. To perform a measurement of the distance between two points, one simply fastens the tape on one point with a mouse click, then pulls out the tape until it reaches the second point, where it can be attached with another click. One then has a line segment connecting the points, and the line has a scale, with respect to some unit, from which one can read off an estimate of the distance between the points. The computer can also supply a more accurate numerical reading of the distance. The scale on the ruler can be changed at will by adjusting the unit. Students will notice that this kind of rescaling is different from the one using manipulation tools. This one keeps the size of objects fixed while the size of the unit is changed. The other resizes the objects while the unit is fixed. The latter is called an *active transformation,* while the former is said to be *passive.*

Another useful type of ruler is a *folding ruler,* which works much like the tape measure, except that it is composed of straight line links of fixed length connected by freely bendable joints. This kind of ruler can be used for polygonal approximations to curves.

Our protractor tool is simply a unit circle with a scale (in radians or degrees) which can be centered at any point on the screen to compare directions there. The circle displays a horizontal radius vector which serves as a *reference direction.* This vector does not change direction when the protractor is displaced. In other words, all displacements of the protractor are translations, accompanied by *parallel transfer* of the reference vector to the point where the protractor is centered. This makes it possible to compare directions at different points. Parallel transfer is one of the fundamental concepts of higher geometry incorporated into the design of our tools.

Our measuring tools will also be able to introduce coordinate systems. A coordinate system can be regarded as a generalized *ruler-protractor*, because it combines the capabilities of both. Our coordinate system tool allows many options, including

origin	
frame	
axes	rectangular
coordinates	polar
	skew

Selection of any of these options immediately instantiates a depiction on the screen which reveals precisely what the option entails. The display will also be manipulatable. Actually, we have already created and used a coordinate tool with many of the options we need.

After the kinematics kit has been developed, depictions of *radar* and *sonar* measuring tools can be built. Such tools are essential for measuring distance in relativity theory.

Although we have restricted our discussion to 2-d geometry, we will also develop the generalization to 3-d geometry along with some new tools as well. The first task will be to model the 3-d geometry of simple rigid bodies, such as spheres and cubes. Beyond that, we will create coupled formal and depictive representations of *Frenet frames* for arbitrary differentiable curves, including a depiction of the "osculating circle" as a point slides along the curve. Ultimately, we intend also to develop a complete kit of tools for characterizing the differential geometry of 2-d surfaces.

Kinematics tool kits

Kinematics is the geometry of motion, so kinematics tool kits are extensions of our geometry tool kits. In accordance with our general plan, we are designing self-contained tool kits of increasing sophistication from 2-d through 3-d to 4-d (spacetime) descriptions of motion. Einstein's theory of special relativity is fundamentally a kinematic theory; he regarded it as the natural completion of ideas about space and time which developed over centuries. We believe that our tools will make relativity more obvious, understandable and accessible to students at an earlier stage in the physics curriculum.

Our kinematics kits are of two main types: Kits for describing, depicting and manipulating models of (a) particle motions and (b) reference frames. We discuss both in turn. Relativity requires a modification of reference frame kits to codify the operational concept of temporal synchronization. We will omit that from our present discussion for lack of space and time.

The kinematics tool menu will include the following options:

I. Create (destroy) particle
  - Describe (edit)

II. Create trajectory
  - Draw
  - Import
  - Describe
  - Select
  - Generate

III . Depict motion
- Motion map
  - kinematic fields
  - hodograph
- Simulation
- Graph
- Phase portrait

Next we briefly describe some important features of these modeling tools.

To create a particle (model) when the tool is activated, one merely selects a place on the screen for its "position" and double clicks. Then a spot appears to depict the particle at that position and a pop up *description dialog box* appears as shown in

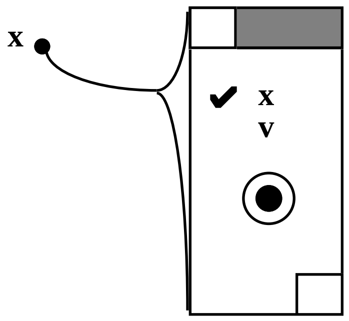


Fig. 2.41.

The box displays a list of default symbols for descriptive variables (descriptors). The list can be edited to replace the defaults with other symbols in the usual way. In fact that must be done if there are two particles on the screen, because no two particles can have the same descriptors. By the device explained before, specification of descriptors automatically hot-links them to all depictions of particle behavior. The check next to the **x** makes it appear as a label on the screen, as shown. Double clicks on **x** and **v** will make values for the position and velocity vectors appear on the screen in proper relation to the point (more below).

The dialog box in Fig. 2.41 has standard Mac features: The box can be moved around by "grabbing" the bar at the top; however, it will remain connected to the original point by the "thread" shown in the figure. The box can be closed by double clicking the square in the upper left, and it can be resized to make room for more descriptors by "pulling" the square in the lower right. Finally, a double click in the button at the bottom will open a more elaborate *description window* which will supply all relevant information about the particle, such as the following details about *descriptors*:

			special values	
Verbal	Formal	Functional	initial	other
name	(label)			
position	$\mathbf{x}$	$\mathbf{x}(t)$	$\mathbf{x}_0$	$\mathbf{x}_k = \mathbf{x}(t_k)$
velocity	$\mathbf{v}$	$\mathbf{v}(t)$	$\mathbf{v}_0$	
acceleration	$\mathbf{a}$	$\mathbf{a}(t)$	$\mathbf{a}_0$	
speed	$v = \lvert\mathbf{v}\rvert$	$v(t)$	v_0	
path length	s	$s(t)$	s_0	
mass	m			
charge	q			

The descriptors "mass and charge" are included to indicate how the description will be extended when one goes beyond kinematic models to dynamics.

We always use vectors for our primary kinematic descriptors. The option to use coordinates will also be available, but that will require specifying a reference frame (as explained below).

After suitable descriptors for the particle have been chosen, a trajectory must be created (i.e. specified). Five different ways to do that with our modeling tools were listed above. Now we describe each one briefly in turn.

(1) *Draw*. This tool enables the user to create (draw) an arbitrary trajectory simply by dragging the particle on the screen from one point to another. The path can then be made into a trajectory by " ticking" (or marking) positions on the path at successive time intervals. In this way the user creates "fictitious data" from which, by smooth interpolation, the computer automatically creates a complete vector-valued function **x**(*t*) for the trajectory. By differentiation it also creates velocity and acceleration functions : **v**(*t*) and **a**(*t*). All trajectories created in this way are smooth. In the dynamics kit, a momentum conservation tool makes it possible to introduce discontinuities in the path (attributed to collisions).

(2) *Import*. This tool is similar to the draw tool, except that the trajectory is constructed from data imported from some external source (such as an experiment). Consequently, the tool must include scaling capabilities to properly match the data to screen parameters.

(3) *Describe*. This tool enables one to create the trajectory function **x**(*t*) from a menu of mathematical functions.

(4) *Select*. One simply selects the desired **x**(*t*) from the "stockroom" of particle models.

(5) *Generate*. This tool obtains **x**(*t*) by integrating a specified differential equation. Of course, it is the job of dynamics to determine that equation.

After a trajectory has been created, it must be depicted in ways which make its structure visible and obvious to the user. Several devices for doing that were listed above. Among those, we think the motion map is the most fundamental, because of its close tie to the geometry of physical space and visual experience. The motion map is a proven cognitive and instructional tool (Hestenes, 1987). The computer makes it even more effective.

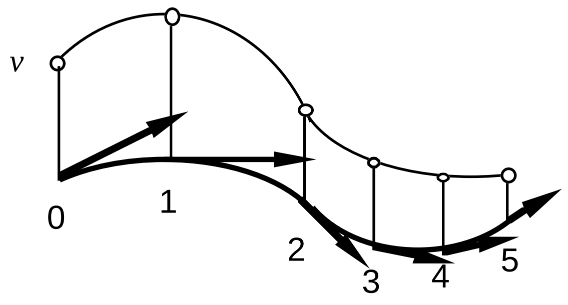
A *motion map* depicts motion by representing kinematic variables as vector/scalar fields on the particle path. For example Fig. 2.42a depicts particle velocity **v** and *speed v* as, respectively, vector and scalar fields on the path, with instantaneous values sampled at equal time intervals. This is an unusual choice of variables, but it serves to make a point. Note that the lollipop values for the speed depicted in the figure can be interpolated to show speed as a continuous scalar field on the path, or, if you will, a graph of speed versus distance traveled. This is easy to map visually onto the more ordinary graph of speed versus time shown in Fig. 2.42b. It should not take much experience with the easily manipulatable computer representations of this type for students to clearly distinguish speed from velocity and graphs from maps. These have been objects of perennial student confusion in the past.

The concepts of scalar and vector fields are fundamental to physical understanding, but usually they are first introduced in the study of electricity and magnetism. There the fields are two and three dimensional, and students have much difficulty understanding them. This difficulty would be considerably alleviated if students first became thoroughly familiar with scalar and vector fields on curves before considering 3-d fields. Our tools will be the first to do this systematically in kinematics.

Kinematical simulations are closely related to motion maps. The main difference is that the motion map represents the entire motion in a single spatial display, while a simulation depicts it as unfolding with time. The advantages of both can be combined by having the simulation generate the motion map as a *trace* (or track) of the moving particle.

For depicting simulations and many other aspects of physics we build the concept of *tangent space* systematically into the design of our modeling tools. "Tangent space" is one of the fundamental concepts of differential geometry, but it is usually introduced only in advanced mathematics. The powerful depictive capabilities of computer modeling tools enable us to take advantage of the concept at a much lower level. The tangent space of a geometrical point **x** is a vector space "attached" to the point. We depict is as a vector space with its origin at **x**, as shown

Fig. 2.42a



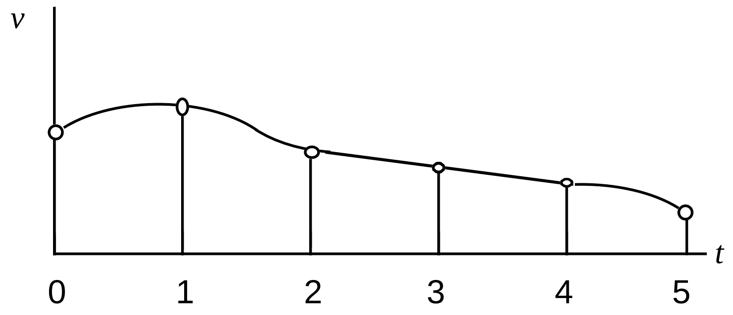


Fig. 2.42b

in Fig. 2.43, where **x** is the position of a particle. The velocity and acceleration vectors are "in" the tangent space, which is depicted by the *dotted window* centered at **x** in the figure. (Note the difference between the *tangent space* at a point and a *tangent line* through the point.) In a simulation, the tangent space moves with the curve. To an imaginary observer riding in this space, the vectors **v** and **a** will be "seen" as changing with time. Our simulation tools can make this literal as a visual image by keeping the tangent space fixed and moving the curve with time. In this representation the velocity traces out a curve **v** = **v**(t) called the *hodograph* of the motion. The hodograph is an underutilized concept in mechanics (Hestenes, 1986). Our tools take advantage of it.

We incorporate the tangent space into a general modeling tool that we call the *tangent window*. As suggested by Fig. 2.44, the window can be moved along a particle path (or any other curve) by dragging its center with the mouse, and it can be preset to display and/or label the

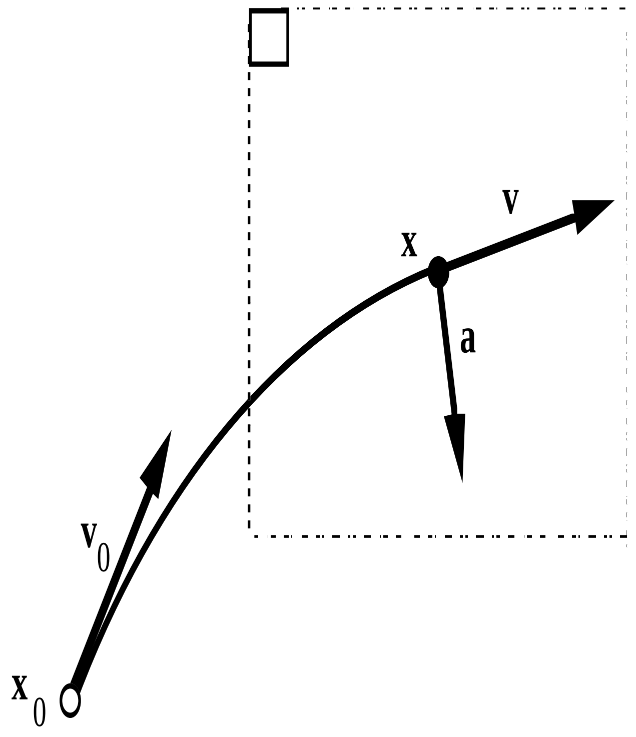


Fig. 2.43. Depicting the tangent space of the point **x**.

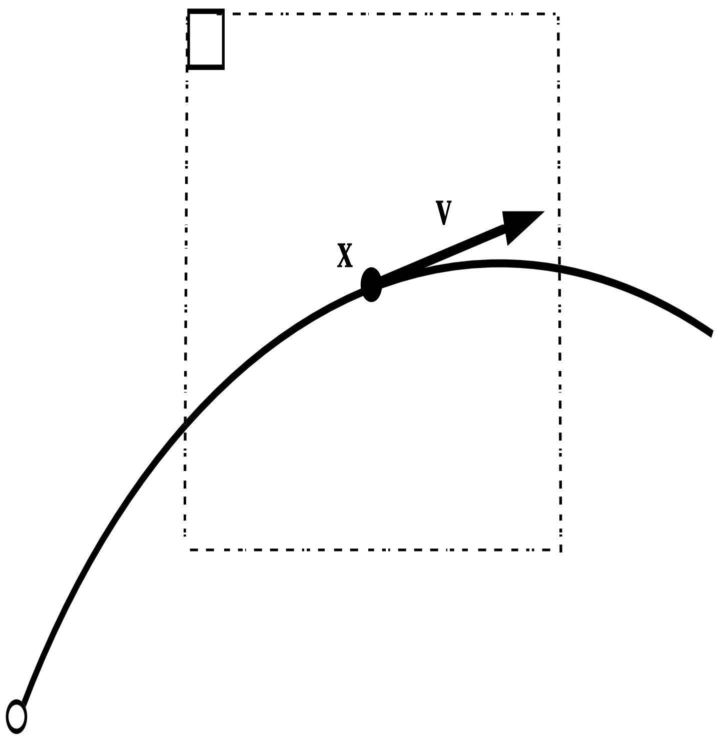


Fig. 2.44 Tangent window.

values of any desired tangent vectors (or scalars) at the point where it is centered. A double click will leave the vector there after the window is removed. With this tool, tangent fields can be very quickly displayed as desired on a screen. By affixing the tangent window to the particle in a simulation, we then get the depiction shown in Fig. 2.43.

Alternatively, if the **v** in the window of Fig. 2.44 is the velocity at the point, then it can be regarded as a parameter of the curve, so if it is varied using the mouse, the curve can be programmed to change accordingly. This should be a *very* efficient for *qualitative curve fitting*. Thus, the tangent window can be developed into a useful tool for *tangent space control* of curves through a given point.

Reference frame

Kinematic tools introduce a crucial semantic distinction into the depiction of spatial relations which is absent from geometry. In geometric modeling, the computer screen serves as a representation of *physical space* modeled as a unique 3-d or 2-d Euclidean space. In kinematic modeling, however, the screen display is depiction with respect to a particular reference frame, and it differs for different frames. The physics of reference frames has long been one of the most difficult topics to teach, in large part, we believe, because traditional depictive tools are clumsy and static, placing an excessive mental visualization burden on the student. Reference frame tools hold great promise for rectifying that.

Like other objects we model, a reference frame has both a description and a depiction which must be set up when the frame is created. Like the "create particle tool," a "create frame" tool opens up a description box where all the relevant descriptors are easily specified. These include (a) a name for the frame: *primed* or *unprimed* (to use a common convention in physics), (b) an origin designated by the zero vector: **0** or **0**', (c) a set of unit vectors: {$\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$} or {$\mathbf{e}_1'$, $\mathbf{e}_2'$, $\mathbf{e}_3'$}. The unprimed frame is a *default frame*, always present, but invisible on the screen unless a "show frame" command activates its depiction. "Reference frames" are generalizations of "coordinate systems," so all the coordinate system tools are included in the reference frame tool box. In addition to spatial coordinates, the frames have clocks which may need to be synchronized. This is critical in relativity theory. All kinematic variables introduced earlier are defined with respect to the default reference frame. Any variable or equation can be redefined with respect to a new frame simply by highlighting it and applying the "change of frame" tool. This tool requires specification of equations for the transformation from one frame to another. For that purpose we have a *displacement operator*, a variant of the manipulation tools described earlier. This operator generates rigid displacements, a combination of translation and rotation. Unlike before, these displacements can be time varying to describe and depict rotating and

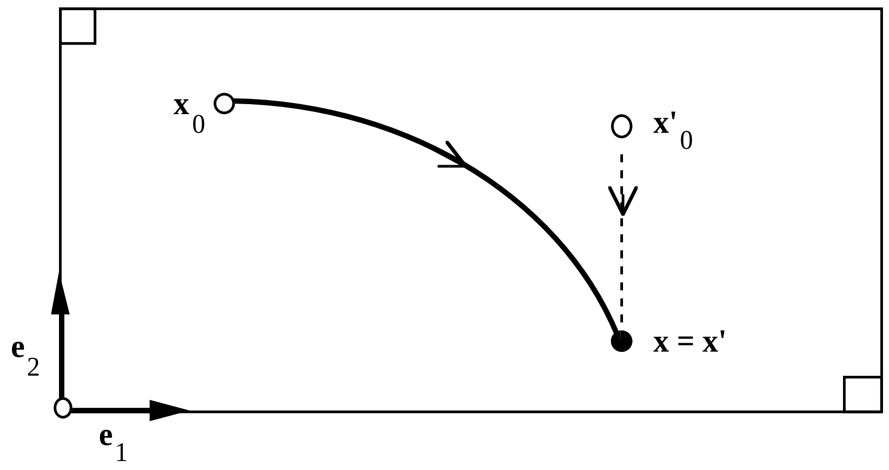


Fig. 2.45. Frame window.

generally accelerating frames. Of course, for transformations between *inertial reference frames* the time variation is highly restricted. The displacement operator tool will be used later in Workstation development for rigid body kinematics.

Our main frame depiction tool is the *frame window* illustrated in Fig. 2.45. It has many capabilities of the tangent window discussed earlier, along with all the coordinate system options. The displacement operator can set the window in motion, or the window can be held fixed while the "background" frame on the screen is moved. Of particular interest is the fact that the window is transparent, so simulation paths generated simultaneously in two different frames can be directly viewed and compared by the user. This will surely be a very powerful tool for helping students understand relative motion.

Dynamics

Dynamics goes beyond kinematics in modeling "outside influences" on the motion of an object. Therefore, a dynamics kit must contain tools for modeling the environment and its interactions with the object. All this is done within the context of some physical theory such as Newtonian mechanics, as was explained in Section 2.3. A systematic classification and analysis of interactions in mechanics is given in (Hestenes 1986, 1987 and 1992). This will be built into our design of "*interaction tools.*" No need to elaborate here.

Dynamical Systems

Dynamical systems theory is best regarded as a branch of general *systems theory* concerned with abstract process models governed by differential equations. It is an abstraction and generalization of Newtonian mechanics, separating form from content, to be used for modeling any process of change in quantifiable property variables. Most of the modeling tools we have discussed can be accordingly generalized for modeling general dynamical systems. This broadens the scope for applications of our modeling tools enormously, so we will provide for it in our designs.

Math Utilities

At the higher levels of sophistication, the Modeling Workstation will have a full complement of mathematical utilities for use in modeling. Some powerful mathematics packages, such as *Mathematica*, *Maple* and *Mathcad*, are now available and keep improving. The Workstation must be designed to take advantage of these resources. However, none of the math packages meets our criteria for interface design, so we must design our own interface and call them up as subroutines.

The most crucial failing of math software and mathematics in general is an inadequate treatment of semantics, a critical issue in modeling. It may be said that *in the systematic treatment* *of semantics modeling theory goes beyond mathematics.* True, there is a branch of formal mathematics called "model theory," but its obscurity is symptomatic of the fact that it falls short of meeting the needs for modeling in science.

The semantic weakness of mathematics is evident in treatments of the concept of *function*, rightly regarded by many as the central concept of mathematics. Functions appear under many pseudonyms, including

(1) operators,
(2) transformations,
(3) fields,
(4) change of representation.

All these terms are commonly used by mathematicians without specifying precisely what distinguishes them. Everyone recognizes that the universal concept of function embraces them all, and that any of the terms might be applied to one and the same functional form. Usage is dictated informally by custom and context. However, the very consistency of the usage suggests something deeper. We submit that consistent use of pseudonyms for "function" is governed mainly, though not exclusively, by *tacit semantic knowledge*, and we take it as a job for modeling theory to make that semantic knowledge *explicit*, so it can be systematized like the rest of mathematics. As our discussion in earlier subsections suggests, a promising way to proceed is by formalizing depictive syntax and its relation to syntactic syntax. We have started that in our design of modeling tools. Specifically, we distinguish sharply between *fields* and *transformations* by the depictions we give them. A given vector-valued function of a vector variable could be either a field or a transformation. For a field, the range and the domain of the function are of semantically different types, and this is expressed in its depiction by arrows in tangent spaces. For a transformation, on the other hand, the range and domain are of the same semantic type. In truth, several different types of transformation can be distinguished by different semantics, but this is not the place for details. Semantics, as a formal discipline, is still in its infancy. The message here is that computer depiction tools may help it grow up.

2.5. Modeling Systems

In this Section we describe the organization of Modeling Systems software at the highest level in the Workstation. The entire system is designed to reflect the structure of scientific knowledge in general and of physics in particular. This includes both factual and procedural knowledge. Procedural knowledge is built into the design of modeling tools and into the coordination of tool use required by theories. This knowledge can then be used to analyze and store factual knowledge so it can easily be retrieved and applied when needed.

Two interfaces, two metaphors

To express the division of science into empirical and theoretical components, we create a separate software subsystem for each. Each subsystem has its own interface, though both employ the same computer screen. Figure 2.51 is a global schematic of the Workstation showing interaction with the user through empirical and theoretical interfaces along with links to external sources of information.

The *empirical interface* displays empirical data/information which has been imported either directly through connections to scientific instruments or indirectly by video tape or by some other means. We construe the term "empirical" to embrace both observation and experiment. The term "experimental" implies some manipulation of natural processes, whereas "observational" (as in astronomy) does not. The empirical interface may provide some capabilities for acting on the physical world, for example, by controlling apparatus in an experiment, but the data gathering and display functions will be dominant. Of course, the user can select and manipulate whatever data is of interest by acting on the interface.

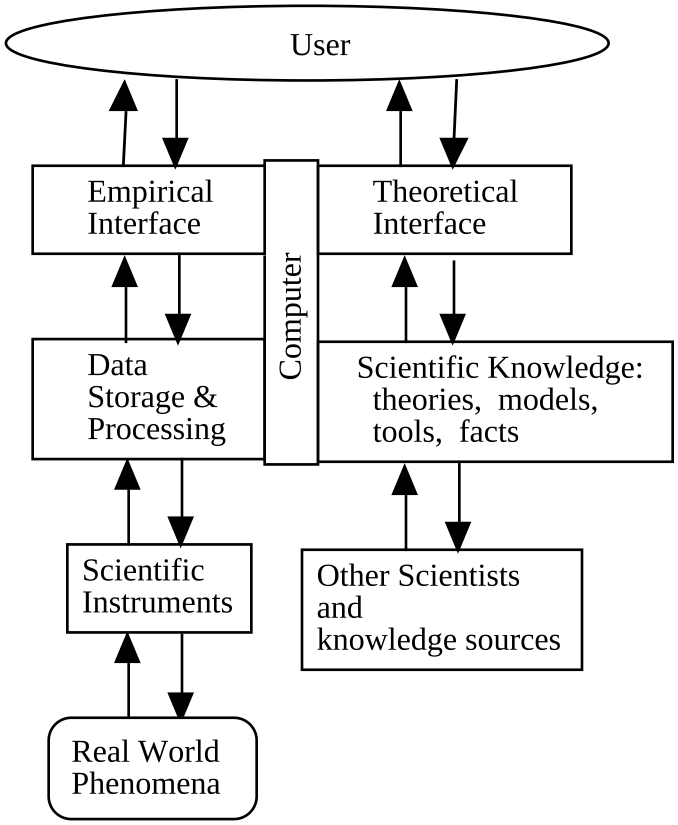

User
Empirical Interface
Theoretical Interface
Computer
Data Storage & Processing
Scientific Knowledge: theories, models, tools, facts
Scientific Instruments
Other Scientists and knowledge sources
Real World Phenomena

Figure 2.51. Modeling Workstation: General structure and external links.

The *theoretical interface* has all the tools needed to construct and analyze scientific models, including coupled formal and depictive representations, in given theoretical domains. It also provides easy access to standard models and factual knowledge useful for modeling in those domains.

We have noted the importance of metaphors to help users understand the structure and utility of software systems. Our choice names for the two interfaces is intended to support a *social metaphor* wherein the Modeling Workstation embed the user in the scientific community. The empirical interface is thus a metaphor for *shared experience* of the real world viewed through the eyes of scientific instruments. The experience is shared in the sense that the interface presents the same view to anyone who inspects it. Similarly, the theoretical interface is a metaphor for *shared knowledge*, for through this interface everyone has equal access to the common body of scientific knowledge. Such shared experience and knowledge is the *foundation for objectivity* in science. The Workstation creates a universal, reproducible *scientific environment* to which every user has equal access. To a group of students using the same Workstation, the scientific environment provides the rich body of common experience needed to provoke meaningful scientific dialog among the students. How the teacher should deploy the Workstation to best achieve such a result is a matter for future research.

Besides the social metaphor for Workstation function, the *brain metaphor* schematized in Fig. 2.52 may be enlightening. In this metaphor we replace the terms "empirical and theoretical" with *perceptual* and *conceptual* respectively, and, for historical reasons, we call the user Homunculus. The Workstation is then seen as an expansion of the human brain, increasing the mental powers of Homunculus. Through the *Perceptual Interface* Homunculus gains a view of the world from scientific instruments that far surpasses the human senses in sensitivity and range. Through the *Cognitive Interface* Homunculus engages the power of scientific theory to augment

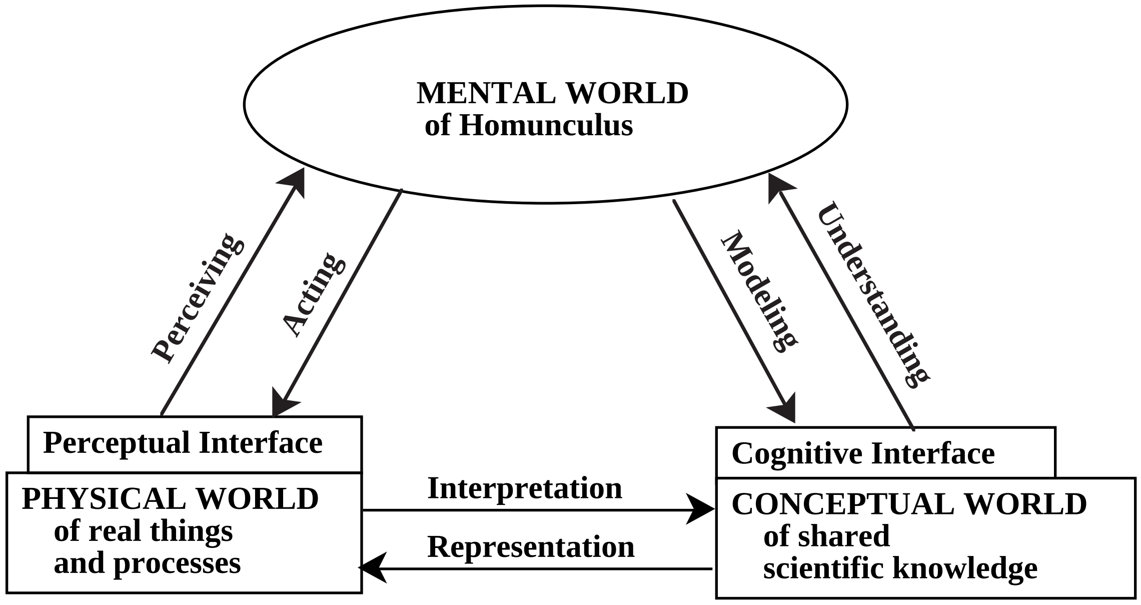

MENTAL WORLD of Homunculus
Perceiving
Acting
Modeling
Understanding
Perceptual Interface
PHYSICAL WORLD of real things and processes
Interpretation
Representation
Cognitive Interface
CONCEPTUAL WORLD of shared scientific knowledge

Figure 2.52. Brain metaphor for Modeling Workstation function.

his thinking. Insofar as students see the Workstation as enhancing their powers of perception and cognition, they will recognize *science as empowering*!

There is an inherent epistemology in our subdivision of scientific knowledge into empirical/perceptual and theoretical/cognitive components. On this matter Fig. 2.52 should be compared with Figure 5 in Hestenes (1992) and its discussion. This is not the place for a philosophical analysis.

Empirical/Perceptual Interface

The perceptual interface is composed of windows for displaying and correlating different types of data, organized into three processing levels, as follows:

- I. Raw Data
  - (A) Image Window
    - Visual
    - Audio
  - (B) Instrument Panel
- II. Data Display
  - (A) Graphical
  - (B) Tabular
- III. Processed Data
  - (A) Statistical analysis
  - (B) Signal/image processing

The *Image Window* presents sensory data for direct observation and analysis by the user. It is capable of presenting both visual and audio images. The visual images may be digitized videos, animations of simulations from any standard source: camera, CD Rom., computer programs, telecommunication. The Window will have a tool kit for manipulating images and extracting data, such as measurements of speed and acceleration, for analysis.

The *instrument panel* is a window for displaying measured values from scientific instruments, such as an odometer, a clock or a thermometer. A menu will provide options for customizing the display, with choices of digital or analog displays and instrument icons suggesting physical interpretations for the data.

At the second level of processing, raw data from the image and instrument windows is organized in Data Display windows for reference and analysis. *Graphical windows* are equipped with tools for rescaling variables and otherwise manipulating graphical displays. In fact, we have already completed the development and testing of a very powerful and flexible graphical window as part of our MBL software (Thornton, 1989). For modeling purposes, our window is superior to commercial graphing software. For example, it is capable of generating graphs in *real time* from data collected by MBL sensors.

Our graphical window is linked to a *tabular window* for an alternate representation of data in spread sheet form. In fact, all windows in the perceptual interface are coupled to support coordinated multiple representations of data, including real time concurrent display of several variables when appropriate. In particular, the real time generation of graphs can be synchronized with visual images. This clarifies the semantic meaning of the graphs. Similarly, it is often of interest to display an audio image synchronously with its sound spectrograph. The computer is blessed with perfect memory, so all temporal records can be reviewed and analyzed at any convenient speed.

The third level of processing includes tools for statistical analysis and signal/image processing at various levels of sophistication. This level is not needed for elementary applications of the Workstation.

Theoretical/Cognitive Interface

The theoretical interface gives the user *access and control* of a vast quantity of scientific knowledge codified in Workstation software. The design problem of concern in this subsection is the construction and manipulation of interface representations to make the structure of that knowledge as self-evident and accessible as possible. The design of modeling tools to use the knowledge effectively was discussed in Sec. 2.4.. The structure of knowledge is critically dependent on the available tools, so a sharp separation of structure from tools is not to be expected. We make the separation only for methodological convenience. In fact, the best way of characterizing structure may be as a system of general tools for coordinating the use of specific modeling tools.

Our first design task is to clearly distinguish *generic structure* common to all physical theories from *specific structure* peculiar to particular theories and make this distinction obvious and natural in the interface. The reason for this is that generic structure unifies scientific knowledge and is transferable from one domain to another. There is an extensive field of research on generic structure called *systems theory*. We will embed the insights of systems theory into our interface design. We will design generic tools for characterizing the structure and behavior of any physical system and supply the means for incorporating it into the structure of specific models.

To cite an example which we intend eventually to work out in full detail: Electrical circuits and heat engines are standard subjects in University physics. They are accompanied by elaborate diagrams to help students understand how they work. What is not at all clear either from the diagrams or the texts that go with them is that these subjects share a common central theme – the storage and flow of energy. We will create representational tools that apply to both subjects and so make their common elements explicit. In fact, these tools will include a variant of the tool for representing scalar fields on particle trajectories, as discussed in section 2.4.

The generic structure of systems theory is reflected in the specific structure of scientific theories. Physics consists of a network of overlapping theories, and students will discover this by opening a THEORY MENU and exploring the contents. The selections available on the menu include NEWTONIAN theory, ELECTROMAGNETIC theory and RELATIVITY theory. Each theory defines a conceptual world which students can select with a mouse-click. For example, the NEWTONIAN WORLD is defined by the system of laws displayed schematically in a *dialog box* which appears when the Newtonian theory is selected (Fig. 2.53, adapted from Hestenes 1992). Note that the laws fall into three general categories which apply to every theory:

Fig. 2.53.

THEORY DIALOG BOX OK
Cancel

NEWTONIAN THEORY DEFINES THE NEWTONIAN WORLD.

ENTITIES: Particles and Bodies = {systems of particles}

KINEMATICAL LAWS.

0th Every particle *k* has a definite *position* $\mathbf{x}_k$
with respect to a given *reference frame.*

Motion of the particle is represented by a
trajectory $\mathbf{x}_k(t)$.

1st An *inertial system* is a reference system in which
every *free particle* has a constant velocity.

DYNAMICAL LAW

2nd In an inertial system, $\mathbf{F}_k = m_k \dfrac{d^2\mathbf{x}_k}{dt^2}$

INTERACTION LAWS

3rd $\mathbf{F}_{12} = -\mathbf{F}_{21}$

4th $\mathbf{F}_k = \sum_{j=1}^{N} \mathbf{F}_{kj}$

5th $\mathbf{F}_{12} = \mathbf{F}(\mathbf{x}_1 - \mathbf{x}_2, \mathbf{v}_1 - \mathbf{v}_2)$

kinematics, dynamics and interactions. For a detailed discussion of any law (following Hestenes, 1986, Chap. 9), the student clicks on the appropriate LAW button. The ZEROth LAW, defining the concepts of *space* and *time*, is the most complex of the laws and the most fundamental, because it is the foundation for measurement theory. Ironically, it is not even identified as a law in standard textbooks. That is not a minor oversight, for the shift from Newtonian mechanics to Relativistic mechanics is the consequence of a subtle alteration of the Zeroth law. Students are able to discover such differences among physical theories by exploring the Theory Menu.

Most of the time the student will not be interested in exploring the structure of the selected theory, but wishes to take the theory for granted. By clicking the OK button in the upper right hand corner of the dialog box, the student accepts the Newtonian World and the box disappears, clearing Desktop for modeling activities. The theory remains active behind the scenes, however, by constraining what can be extracted from the other menus to conform to Newtonian theory. Hence there is a MODEL MENU offering the student a selection of Newtonian MODEL TYPES, including PARTICLE, RIGID BODY, ELASTIC SOLID, IDEAL FLUID, IDEAL GAS. A mouse-click on one of these starts the student on a path through the stages of model development (Hestenes 1987) guided by a sequence of dialog boxes and forced choices. It begins with a *dialog box* asking the student to specify a set of OBJECT VARIABLES such as mass and charge for a particle model or geometrical shape and moments of inertia for a rigid body. TOOLS for constructing suitable graphical, diagrammatic and symbolic representations are available, along with prompts to help students learn to use them effectively. An INTERACTION MENU offers a classification of available SPECIFIC FORCE LAWS, including both phenomenological and fundamental types, along with ancillary information about them. Among other things, a PREFERENCE MENU enables the student to switch from a force to a potential energy representation of interactions. To produce equations of motion for model objects, the student must substitute suitable forces into the dynamical laws.

Modeling Task Manager

To coordinate the Empirical and Theoretical Interfaces and to manage complex Workstation activities, a third interface is needed, the *Modeling Task Manager* (MTM). For this purpose, we need a systematic classification and analysis of modeling tasks to explicate and systematize strategic and procedural knowledge involved. The highest objective of this project is to incorporate such knowledge into the design of the MTM, where it will be of practical value to users. By way of example, we describe two important activities that the MTM will be designed to facilitate: semantic matching and recording/reporting.

Semantic matching

To span the semantic gap between theory and observation, we need tools that compare theoretical models with observational data. The simplest kind of quantitative semantic matching is *curve fitting.* Computer tools greatly facilitate this activity. On a window in the Theoretical Interface a curve generated by a parametrized model can be displayed. Let this window be *transparent,* so it can be overlaid on a data window in the Empirical Interface. The user then varies the parameters to "eyeball" a best fit of the curve (hence, the model) to the data. The "semantic matching tool" is designed to make obvious the role of *units* in the scaling required to compare the model with data. The tool can be refined for quantitative matching using statistical tools.

In a dynamic form of semantic matching, a model generated simulation of particle motion is overlaid on a video of the object being modeled.

Recording/Reporting

The computer has an unmatched capability to record actions of the user in modeling activities. Kaput (1992) notes two complementary types of record: (a) Records of prior states called *filmstrips*, and (b) records of actions (that produced the state changes) called *scripts*. The user can replay, analyze and *edit* the records to perfect the activity of interest. This could be anything from a calculation to a proof of a theorem. As Kaput notes, editing a script is a kind of "programming by doing." For repeated applications, a script can be assigned a command to execute it as a single step.

Records are seldom of more than passing interest unless they are embedded into reports, notebooks or portfolios which put them in a significant context. For constructing such reports, the MTM provides access to general utilities such as a word processing system. In addition to ordinary word processing, we plan to make available an interactive notebook that can function (1) as an electronic curriculum guide for guided inquiry, (2) as a means of presenting tutorials, including those introducing students to the workstation environment, and (3) as a means for students to communicate findings to other students and to teachers. For example, when used for guided curriculum, the notebook could include text, graphics, questions with space for the student to type an answer or draw a prediction, and interfaces to the modeling tools. Tools would be started and automatically configured for the current use right from the notebook, thus acting as extensions of the notebook environment.

III. Toward an integrated software curriculum for all grade levels

The current computer revolution presents an unprecedented opportunity for curriculum reform in science and mathematics. Computer infusion into classrooms is sure to continue at a rapid rate, driven by steadily improving hardware at decreasing cost and by the hope for a technological fix of educational problems. This trend is creating a "software vacuum" which will be filled with "junk" software unless a clearly superior "software curriculum" is developed to displace it. Since junk software does not address fundamental pedagogical issues, it will exacerbate educational problems by diverting attention and resources. The result will be further disillusionment with education and retrenchment in public support.

This impending software crisis can be averted if scientists, educators and software developers collaborate on the development of a *integrated math-science software curriculum* which is scientifically and pedagogically sound. We say "software curriculum" rather than "curriculum software," because we see it as an agent for curriculum reform rather than an enhancement of the standard curriculum. The aim is to build the curriculum across all grade levels into an integrated system of software packages. This approach to curriculum reform is potentially far more effective than the mere publication of authoritative curriculum standards and recommendations, because software design can be more strictly controlled, and, riding the wave of technological change, software can more easily penetrate bureaucratic and academic barriers standing in the way of reform.

Little curriculum reform can be expected from stand-alone software packages, no matter how brilliantly conceived. Without a software curriculum to guide adoptions, naive users, from teachers to school boards, will be unable to distinguish between quality and junk software. Junk software will be attractively packaged and vigorously marketed by commercial vendors. Most will be offshoots of available commercial software rather than designed to satisfy well considered pedagogical criteria. Beyond the difficulty of getting adopted in the schools, there is a more serious limitation to stand-alone software.

The *fragmentation of knowledge* acquired by students is arguably the most serious failing of the current math-science curriculum (di Sessa, 1988). Software packages designed for coherent treatment of individual topics or courses will not suffice to rectify the problem. Competence in science and mathematics involves complex cognitive skills requiring many years and broad experience to develop. Therefore, an integrated software curriculum is needed to foster math-science competence efficiently. At the very least, general agreement on conventions and standards for human-computer interface design is needed to minimize the cognitive overhead needed to move from one software package to another. Even more can be achieved through cooperative research and development.

The software curriculum needs *vertical integration* across grade levels and *horizontal integration* across subjects and disciplines. "Graded" software packages can be designed to promote student cognitive development along a progression of competence levels. Competence standards can be built into the graded software, so that proficiency with the software is a clear measure of competence, and each software package contains clear specifications of prerequisite knowledge.

Horizontal integration helps students see how powerful conceptual structures apply across a variety of situations. Such experience is essential to promote knowledge transfer across the boundaries of scientific domains. Indeed, the surest defense against befuddlement and despair from the current information explosion is an appreciation of the coherence of scientific knowledge.

An integrated math-science software curriculum cannot be achieved without a broad consensus among its developers on design principles, guidelines and specifications which promote integration without limiting the creativity of individual developers. The software designs must be grounded in a coherent theory of scientific knowledge, including its use and acquisition. We have proposed an approach grounded in *Modeling Theory* which can be summarized in the following **cardinal design principle:**

> ***Designs of math-science instruction as well as software should be centered on condeptual models and modeling*.**

We invite like-minded colleagues to join us in the immense and exciting task of *integrated math-science software design and development.*

Acknowledgements: Thanks to Ibrahim Halloun for help in preparing the manuscript. This work was partially supported by the National Science Foundation grant # RED-9200442.

REFERENCES

J. R. Anderson (1983), *The architecture of cognition.* Cambridge MA: Harvard U. Press.

J. Barwise & J. Etchemendy (1991), Visual Information and Valid Reasoning. In. W. Zimmerman & S. Cunningham, *Visualiqation in Teaching and Learning Mathematics*, Math. Ass. of Am..

A. diSessa (1988) Knowledge in Pieces. In G. Forman and P. Pufall (Eds.) *Constructivism in the Computer Age*, Hillsdale, NJ: Lawrence Erlbaum Associates.

A. diSessa & H. Abelson (1986). Boxer: A reconstructable computer medium. *Communications of the ACM,* **29**, 859-868.

R. Giere (1988), *Explaining Science*, U. Chigo Press.

D. Hestenes (1986), *New Foundations for Classical Mechanics,* G. Reidel Publ. Co., Dordrecht/Boston; paperback 1987, fourth printing 1992.

D. Hestenes, A Modeling Theory of Physics Instruction, *American Journal of Physics* **55**, 440-454 (1987).

D. Hestenes (1992), Modeling Games in the Newtonian World, *American Journal of Physics* **60**, 732-748.

Interactive Physics (1990), San Francisco CA: Knowledge Revolution Inc..

N. Jackiw (designer, 1992), *The Geometer's Sketchpad*, Berkeley, CA: Key Curriculum Press.

G. Lakoff (1987), *Women, Fire and Dangerous Things,* U. Chicago Press.

J. Kaput (1992), Technology and mathematics education. In D. Grouws (Ed.) *Handbook on research in mathematics teaching and learning,* (515-556). New York: Macmillan.

D. A. Norman (1983), Some Observations on Mental Models. In Gentner and Stevens (eds.), *Mental Models*. Hillsdale NJ: Lawrence Erlbaum. p. 7-14.

E. Redish (1994), Implications of cognitive studies for teaching physics, *Am. . Phys.* **62**, 796-803.

J. Schwartz & M. Yerushalmy (designers, 1993), *The Geometric Supersupposer*, Pleasantville NY: Sunburst Communications.

R. Thornton (1989), Tools for scientific thinking: Learning physical concepts with real-time laboratory measurement tools. In E. Redish and J. Risley (Eds.), *Proceedings of the Conference on Computers in Science Teaching*, Reading, MA: Addison Wesley. pp. 177-189.

R. K. Thornton & D. S. Sokoloff (1990), Learning motion concepts using real-time microcomputer-based laboratory tools, *Am. J. Phys*. **58**, 858-867.

M. Wells, D. Hestenes & G. Swackhamer (1994), A MODELING METHOD for high school physics instruction, *The Physics Teacher* (submitted).

B. Y. White (1983), Sources of Difficulty in Understanding Newtonian Dynamics, *Cognitive Science* **7**, *41-65.*