Physical Model Language: Towards a Unified Representation for Continuous and Discrete Models

Matthieu Chabanas and Emmanuel Promayon

Laboratoire TIMC-IMAG-CNRS UMR 5525, Université Joseph Fourier
Institut d'Ingénierie de l'Information de Santé (In3S)
38706 La Tronche cedex, France
{Matthieu.Chabanas,Emmanuel.Promayon}@imag.fr

Abstract. Different approaches exist for modeling human tissues, mostly discrete and continuous physical models, e.g. respectively Mass-Spring Networks and Finite Element Method. Whatever approach is chosen, the modeling scheme always follows the same pattern from the generation of the 3D geometry to the analysis of the simulation results. However there are no generic tools that allow for designing a physical model independently from the approach. This yields to the development of specific tools that are not reusable and that do not facilitate the comparison between methods. In this article we propose a framework that takes into account every step of the modeling process, and that can be used for any type of approach. We define an extensible language to represent both continuous and discrete physical models as well as a language to define constraints and loads to be applied during simulation. The usability of this generic framework is shown through two examples.

1 Introduction

Two different approaches are mainly used to model human soft tissues [1]: continuous approaches, e.g. Finite Element Method (FEM), and discrete approaches, e.g. Mass-Spring Network (MSN). Continuous approaches offer a strong theoretical background but are usually time consuming and not easy to use for dynamic simulations of complex models with interaction between different types of tissues. Computational discrete approaches are often faster than continuous methods and generally offer a way to build complex models. Their main drawback is their lack of parameter control and assessment.

Depending on the required type of simulation, a choice has to be made between the two approaches. Although its implementation depends of the chosen method, the modeling scheme is always organized in four main stages: *a*) the geometry representation; *b*) the properties, parameters or behavior definitions; *c*) the specifications of the constraints and loads; and *d*) the solution representation (in terms of displacements, state changes, ...). Continuous and discrete modeling essentially differ on how the solution is computed from a given set of

S. Cotin and D. Metaxas (Eds.): ISMS 2004, LNCS 3078, pp. 256–266, 2004.

3D geometry, parameters, constraints and loads. For a continuous model a FEM solver is generally used, whereas for a discrete model an animation motor generally provides integration of forces and computes the dynamic changes. Although the transition between stage *a*) and *b*) is where the two approaches differ, the four enumerated stages always have to be completed. But because of the nature of the available modeling tools, all stages are bound to the choice of the approach, thus making it difficult to use generic tools or build generic models that can be tested with different approaches.

In this paper we propose a modeling framework to generically represent a physical model so that these four stages can be untied from the chosen modeling approach. Therefore solvers or animation motors can be used as external tools to generate the simulation, independently from the model construction.

1.1 Goals

A generic modeling architecture has to be able to deal with all kind of approaches: physically-based model, MSN, FEM and even kinematic-only models, regardless of the geometry type (surface or volume).

When dealing with complex physical models, such as found in human modeling, the identification of the simulated objects is crucial. To be generic, a modeling framework has to allow for separation of structural definitions, i.e. geometry and relationships between modeled structures, and structures identifications. This implies that the labelling of the properties and entities can be made at different levels: physiological and functional levels (group of muscles, organs, ...); anatomic levels (name of a particular muscle, type of a particular tissue, ...); but also at general modeling levels (group of "objects", structures of the same types, link between structures,...)

Such a modeling framework has also to offer an extensible serialization format. The documents have to be easily exchanged and modified, and have to support automatic processing for visualization and conversion to or from other platforms.

1.2 Motivations

In the field of geometry representation numerous languages or formats already exist, and some are well established as standards in 2D (SVG) and 3D (VRML or its successor X3D). However good the latter is to represent a 3D scene, the modeled properties are mainly geometric (colors, lighting,...) and the extensive use of a scene graph limits them to 3D graphics applications. ITK and VTK [2] libraries, although they could be used to build a geometry from medical images using powerful image processing and 3D visualization algorithms, are not well suited to represent physical models. More complex visualization and medical image segmentation softwares, freely available such as Julius [3] or commercially distributed such as Analyze [4] or AmiraVis are developed to help a user through an intuitive graphical interface to extract information and build a 3D model from

medical images. Unfortunately they do do not offer a representation of physical structures and their properties.

Modeling softwares or libraries are numerous, especially for FEM modeling. Some object-oriented libraries were developed such as in [5], but they are more geared to build a FEM solver without considering other approaches. Simulation environments have also been developed to help the integration of discrete simulator in graphical application with interactive and tactile tools [6][7]. They are well suited to design simulations, but are not intended to help the comparison between approaches and factorization of the modeling tools. While most of the commercial solvers are extremely powerful and well-tested, they are not specifically designed for human modeling. Thus, they do not provide the environment, data processing and interactions required in medical simulation.

In our research work to date we have not come across any specific framework that has met our needs for a high-level modeling tool that could fill the gap between modeling software and solver or animation motors.

2 Method

A generic framework has to be independent from all modeling methods. Approach-specific terms, such as node, element, vertex, or mass, have to be withdrawn for method-independent terms. In our framework, the 3D representation and definition of structures are distinguished from the labeling and identification of modeled entities. Labeling is allowed for any entity or structure at any level. There is no limit of definition for sub-group or sub-structures. The framework is based on a XML document format defining what we called the *Physical Model Markup Language* (PML). This leads to human-readable documents, that can be automatically processed for conversion or visualization, but also enable us to easily extend the stored information. Advantages are taken from the extensibility feature of XML in order to allow any approach to include specific properties into PML.

On top of PML, a library has been developed using object-oriented techniques. A PML document can thus be bound directly to dynamic objects, allowing flexible representation and manipulation of models. A common architecture is proposed with a common high-level methodology for all types of approach. Specifics, such as the definition of properties and behavior, can then be implemented for each approach using heritage and polymorsphism properties.

2.1 Representing Geometry: Structures

Atoms. Any complex 3D model has to use specified 3D positions, referred to in PML as *atoms*. The equivalence between an atom and other approach entities is detailed in Table 1. An atom is defined by *a*) a position (i.e. 3D coordinates); and *b*) some properties. For example, a mass property could be its mass, expressed in grams, can be assigned to an atom. Atom properties are optional, e.g. in the FEM, there will be no specific atom properties.

Table 1. PML Babel fish table. Equivalence of terms between PML and other representations. Equivalence is only true from PML $\mapsto$ other representation and is not valid in between columns (i.e. a spring in MSN is not equivalent to an element in FEM, but they will be both represented as a cell in PML).

PML	3D Geometry	FEM	Mass-Spring
atom	vertex	node	mass
cell	polygon, polyhedron	element	spring
Structural Component	triangular mesh	FEM mesh	Mass-Spring Network
Multiple Component	several objects or different entities inside an object		

Note that atoms have to be described *independently* from any structure that uses them. Such a structure have to use references to the atoms id. For example in a FEM mesh, the elements that are defined by a given list of atoms will use reference to those atoms id, which will all be described separately).

Cells. Models are usually composed by collections of *cells*, each cell being built with atoms. Cells represent geometrical polyhedron structures (triangles, hexahedrons, ...) present in 3D models (see Table 1 for equivalences). A cell is defined by: *a*) a geometric type: triangle, quad, tetrahedra, hexahedron, line, poly-line, etc...; *b*) a group of atoms used in the cell geometry; and *c*) some associated properties.

While they can be related to visualization, cell properties are mostly used to describe specific physical properties of an approach. For example, in a MSN, stiffness can be assigned to a cell representing a spring, or in the FEM mesh, a rheological property, such as the Young modulus, can be defined for a cell representing an finite element.

Structures. Atoms and cells are the basic components of a physical model, thus are both known as *structures*. Figure 1 presents the object-oriented organization that links the structures.

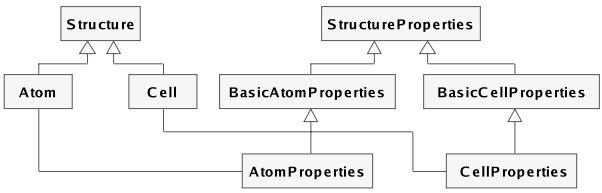


Fig. 1. Partial UML class diagram of the structures (triangles denote inheritance).

Structure Properties. For each type of structure, the associate properties are defined in separate classes. The hierarchical relationship between cell, atom and structure is also applied to their related properties (see Fig. 1). *StructureProperties* holds the common properties (name and id), while *BasicCellProperties* holds

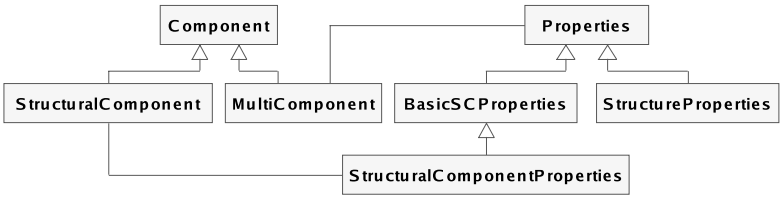


Fig. 2. Partial UML class diagram of the components.

the generic cell properties (geometric type) and *BasicAtomProperties* holds the generic atom properties (position). In order to hold specific properties for a given approach, two classes have to be defined: *CellProperties* and *AtomProperties*. An automatic mechanism allows us to produce these classes from a given set of described properties. Note that even if a document is defined using not implemented specific properties it will still be readable, although these properties will not be controlled.

2.2 Labelling, Representing Exclusive and Informative Meta-structures

For a complex model, it is convenient to be able to group subparts and assign names to these subparts. For instance, it is convenient to have a group containing all the cells that model the muscles, and another one containing all the cells that model fat tissues. It can also be useful for specific representations, to assign properties to a set of structures, to define boundary conditions to another one, and so forth. To do so, the notion of *Component* is introduced in the language (Fig. 2). A component can be either a structural component, or a multi-level component (abbreviated multi-component).

Structural Components. A *structural component* is a group of structures. It is generally composed of cells, although it can be composed of atoms. Note that a cell can also be considered as a structural component as it is defined by a set of atoms.

Multi-level Components. A *multi-level component* (multi-component) consist of any number of components: structural components and/or multi-compo-nents. It is used to group entities and allow hierarchical representation and tree-like organization.

Component Properties. Similarly to structures, the hierarchical relationship of components is applied to the component properties (Fig. 2). A change in a component property is automatically propagated to its sub-components or structures.

Physical Model. Finally, a physical model is defined by three top-level components: the list of all the atoms, the exclusive component, and the informative component.

The **list of all the atoms** is a structural component. All the atoms are defined here; only references to these atoms are used elsewhere in the physical model.

The **exclusive component** is a multi-component. Cells defined here are defined once and only once in this component. No overlapping is possible here. This special multi-component must contain all the cells that are really necessary to the definition of a model, and only those cells. This is the core structure of the physical model.

A multi-component named **informative component** is used to define labeled groups or entities that are not essential for the model but useful for separating or manipulating a given set of structures. The informative components is optional.

2.3 Constraints, Loads and Postprocessing

Once a physical model is expressed in PML, one can use other generic tools to design constraints and loads on the model. In our framework another XML language, called *physical model Loads Markup Language* (LML) is also defined. In LML, a generic load is defined by its targets, its type, its direction, a list of value-events defined in a given unit.

A **target** is the name (or id) of the structure or component, to which the load is applied to.

The **type** can be a single force, a single pressure, a translation, or a rotation (this list can be easily extended to other type of constraints).

The load **direction** is a 3D vector that specifies the direction of the load. This allows us as well to specify null displacement constraints by setting the direction to the null vector.

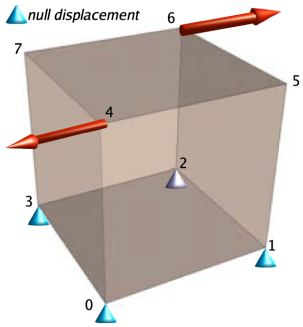
A **value-event** is a pair of numbers: a date and a intensity value. A list of value-events is associated with each load so that it can describe the variation of a given load during time. This allows us to start and/or stop a load during a dynamic simulation, and can also be used to define an intensity profile. A value event can also be used to define the variation type between two value events (square, exponential, ...).

A **unit** is given to a load, to specify in which international unit the intensity values are given.

An example of LML is given in Fig. 3.

2.4 Solver and Animation Motor

The PML plateform is not intended to do any solving or animation. However, it is built to be easily used in conjunction with libraries or solvers by providing them with inputs and by being able to read their outputs, through the use of wrappers. PML and LML documents can be converted as commands or script files for the targeted solver or animation motor. Resulting output data containing the results can then be converted back to LML, e.g. as a list of translations or




```
<loads>
    <load type="Translation">
        <appliedTo>0-3</appliedTo>
        <valueEvent value="0" date="0"/>
        <direction x="0" y="0" z="0"/>
        <unit>nm</unit>
    </load>
    <load type="Force">
        <appliedTo>4</appliedTo>
        <direction x="1" y="0" z="1"/>
        <valueEvent date="0.0" value="50"/>
        <valueEvent date="0.5" value="0"/>
        <unit>kN</unit>
    </load>
    <load type="Force">
        <appliedTo>6</appliedTo>
        <direction x="-1" y="0" z="-1"/>
        <valueEvent date="0.5" value="50"/>
        <unit>kN</unit>
    </load>
</loads>
```

Fig. 3. A LML example set on a simple cubic cell. This cell could be an hexahedron element (FEM approach) or a cube where each vertex is a mass and each edge is a spring (MSN approach). In this example, three loads where imposed: *a*) a null displacement to the base of the cube; *b*) a pulling force on atom 4 starting at at t=0 and ending at t=0.5; and *c*) a pulling force on the opposing atom, atom 6, starting at t=0.5.

strains applied to the nodes. This provides an easy way of storing such results for later observation and animation and for comparison between solutions.

2.5 Implementation

Both PML and LML languages are defined using XML schema, thus allowing for automatic validation and integrity checking as well as easy data-binding with Object-Oriented Programming languages. Two freely available libraries[1], programmed in standard multi-plateform C++ (compatibles with GNU GCC C++ and Microsoft VC++ compilers), allow us to manipulate physical model and loads. Both libraries are using Apache Xerces-C[2] to deal with object serialization and document validation. A plugin for a closed-source application was developed to manipulate PML, demonstrating an easy conversion to 3D using VTK. Different tools and converters for this framework are also freely available.

3 Examples

In this section the usability of PML is demonstrated by two examples, one using a continuous model and the other using a discrete model.

3.1 PML for Continuous Model

PML was used to model the soft tissues of the face for the simulation of maxillofacial surgery using the FEM [8].

[1] open-source GPL license available at `http://www-timc.imag.fr/Emmanuel.Promayon/PML`

[2] `http://xml.apache.org/xerces-c`

Fig. 4. FEM modeling using PML, model of the face. Elements can be grouped by tissue type (left: muscles) or inner/outer layers (center). Nodal components enable us to apply different loads directly to different sets of nodes: null, free or imposed displacements (right).

Structure of the Model. Atoms represent the nodes of the FEM mesh. The *exclusive component* contains a single structural component defining all the cells, which are the finite elements (hexahedrons and wedges).

Organization and Labelling. The *informative* component contains several multi-components grouping elements and nodes at different levels.

Level 1: a multi-component groups the elements by tissue type. It contains two sub-components 'Fat' and 'Muscles' (Fig. 4, left). 'Muscles' itself contains one structural component for each muscle of the model: right & left Major Zygomaticus, right & left Risorius, Orbicularis Oris, ... These structural components simply group references to the cells modeling each muscle.

Level 2: our model is built with two layers of elements, labelled 'Internal Elements' and 'External Elements' (Fig. 4, center). This shows that in the informative components overlaps are allowed: a cell can be simultaneously labelled as being in a muscle and in an external layer.

Level 3: nodes of the mesh are also organized in inner, mid and outer set of nodes, respectively labelled 'Inner Nodes', 'Mid Nodes' and 'Outer Nodes'. Sub levels of the 'Inner nodes' component separate again two groups of nodes: the nodes that are rigidly fixed to facial skeleton (skull base, mandible, maxilla or segments of osteotomy) and the nodes that are free to move (cheeks), Fig. 4, right.

Loads. A LML document contains the different nodal component loads. Boundary conditions consist in null displacements of the nodes fixed to the skull, while the surgical procedure is simulated by an imposed translation applied to some of the the bone components.

Solving. A script file was produced by exporting the geometry definition (nodes, elements), the material properties, and the loads to the Ansys^{TM} Finite Element software (Ansys Inc.). Resulting solutions (node displacements, stress, strain, ...) can then be translated from this solver back into LML in order to animate and analyze the results.

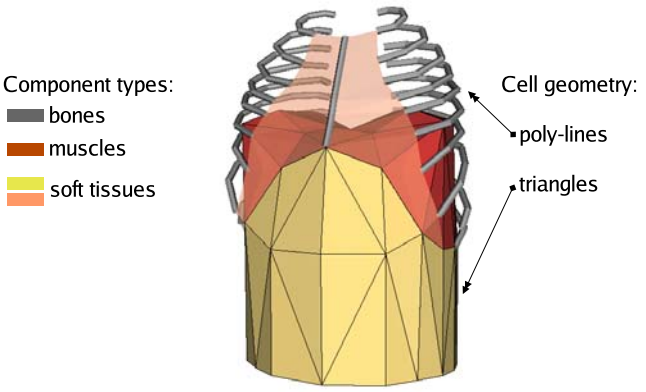


Fig. 5. Discrete modeling using PML, model of the trunk. Muscles are in red, soft tissues in yellow or pink, and skeleton in grey.

3.2 PML for Discrete Model

In [9], an object-oriented discrete model is described where masses can be defined in regions of different types (muscles, soft tissues, skeleton), and where elasticity is modeled using a local shape memory function.

Structure of the Model. PML was used to represent a human trunk and its main anatomic structures. All the masses are represented by an atom. Regions are represented by structural components. The exclusive component contains the list of atoms of each region and the cells defining the atoms neighborhood (these neighborhood are used in this model to compute the local shape memory, and thus the elasticity).

Organization and Labelling. The informative component contains the graphical description of the model (facets representing external surfaces of soft tissues and 'tubes' representing bones), see Fig. 5. Specific properties of this discrete model were added into PML. For instance, custom properties were added to the cells defining the diaphragm muscle in order to store the contraction function parameters.

Loads. Boundary conditions consist in null displacements for masses of the spine and pelvis.

Solving. The animation motor in [9] was modified to take PML and LML documents as input and to write the resulting animation as imposed displacements in LML.

4 Discussion and Conclusion

A generic framework for representing and manipulating physical models has been proposed.

Having a generic architecture allowed us to develop some high-level tools that can be used independently from the chosen modeling method. While this

architecture is based on simple concepts, it must be well understood in order to correctly define a model. This is the main limit of this approach, since the different structures and components do not have the same meaning in all the modeling methods.

On the other hand PML can be used for any kind of geometrically-based model. Meshless methods can be represented only if they are based on some geometry (e.g. it is possible to use this framework to represent the skeleton on which an implicit surface objects is built), but fluids can not be represented per se.

Multiple representations of the same object are also possible. The main representation has to be an exclusive component, all other optional or accessory representations can then be informative components. The volumetric representation of an object and a surfacic representation can coexist, thus making it possible to have a representation used for physical modeling and another one for collision detection and interaction. It is also possible to have, for example, an exclusive component representing the MSN as well as an informative component representing the outlying structure using triangulated surface. Texture properties are not yet represented in PML, but as PML is based on XML, it can be easily extended.

In LML, some basic constraints can be set to a model. This constraints are dynamic as they are strongly associated with value events. Thus it is possible to deal with changing constraints. Contacts are not yet represented in LML.

Generic algorithms, loads, 3D graphics and solution visualization have only to be implemented once whatever approach is chosen. This offer an easy way to compare two approaches and to more easily switch between one approach and another: once the objects are built using the PML framework, modeling and analysis can be automated and the results can thus be straightforwardly compared.

Our proposed framework could be seen as an linking module that can fill the gap between modeling softwares on one side and solvers or animation motors on the other side. For instance it could be integrated as a plugin for a modular software such as Julius or for a simulation and animation plateform.

We are interested in opening a workgroup on comparison of physical model for medical simulation and in discussing the use of PML paradigm with other types of approach.

References

1. Delingette, H.: Towards realistic soft tissue modeling in medical simulation. IEEE: special issue on virtual and augmented reality in medicine **86** (1998) 512–523
2. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit An Object-Oriented Approach To 3D Graphics, 3rd version. Kitware, Inc. (2003)
3. Keeve, E., Jansen, T., von Rymon-Lipinski, B., Burgielski, Z., Hanssen, N., Ritter, L., Lievin, M.: An Open Software Framework for Medical Applications. In: Lecture Notes in Computer Science 2673, proceedings of International Symposium on Surgery Simulation and Soft Tissue Modeling, `http://www.julius.caesar.de` (2003) 302–310

4. Robb, R., Hanson, D.: A Software System for Interactive and Quantitative Visualisation of Multidimensional Biomedical Images. Australian Physical and Engineering Sciences in Medicine **14** (1991) 9–30 `http://www.analyzedirect.com/`.
5. Yu, L., Kumar, A.: An object-oriented modular framework for implementing the finite element method. Computers and Structures **79** (2001) 919–928
6. Meseure, P., Davanne, J., Hilde, L., Lenoir, J., France, L., Triquet, F., Chaillou, C.: A Physically-Based Virtual Environment Dedicated to Surgical Simulation. In: Lecture Notes in Computer Science 2673, proceedings of International Symposium on Surgery Simulation and Soft Tissue Modeling. (2003) 38–47
7. Monserrat, C., López, O., Meier, U., Alcañiz, M., Juan, C., Grau, V.: GeRTiSS: A Generic Multi-model Surgery Simulator. In: Lecture Notes in Computer Science 2673, proceedings of International Symposium on Surgery Simulation and Soft Tissue Modeling. (2003) 59–66
8. Chabanas, M., Luboz, V., Payan, Y.: Patient-specific finite element model of the face soft tissues for computer-assisted maxillofacial surgery. Medical Image Analysis **7** (2003) 131–151
9. Promayon, E., Craighero, S.: Object-oriented discrete modeling: a modular approach for human body simulation. In: International Workshop on Deformable Modeling and Soft Tissues Simulation, Bonn (2001)