# Creation and editing of artifacts' models by Generative Projects

*Antonio Calabrese[1], Carlo Coppola[2], Luca Licenziato[3], Francesco Mele[1], Antonio Sorgente[3], Oliviero Talamo[1]*
*[1]Istituto di Cibernetica "E. Caianiello", C.N.R. - Pozzuoli - Napoli*
*[2]Department of Cultura, Facoltà di Architettura Luigi Vanvitelli, SUN, Aversa*
*[3]Facoltà di Scienze MM.FF.NN - Università di Napoli "Federico II"*
*carlo.coppola@unina2.it*

*In this paper we propose an aiding system for the creation of models of artifacts which is based on a methodology that has its foundations in a concept that we call generative projects. This methodology has been defined separating the design paradigm of the designer from the computational model, defined in order to implement the system that support the designer in the design process, and from the graphical engine of the specific rendering system, chosen for the visualization of the generated artifact. In this work we defined an user interface that assists the designer during the design process, translates the result of the design into the underlying computational model and carries out the access to the rendering system in a transparent way. The experimentation of the system was conducted on various artifacts domains, as jewels, glasses, lamps, cutlery, wireless headphones, aerosols, pots and plans.*

***Keywords:*** *formal ontology; generative design*

## Introduction

The design model we proposed is iterative. In his modelling activity (Coppola et al 2005) the designer moves through three main phases: the hypothesis phase, the model developing phase and the verification phase.

The hypothesis phase is mainly concerned with the model goals and with the beliefs about the spatial relations that can get to such goals.

The model developing phase is mainly concerned with the development of the model and the generation of the 3D-rendering of the artefact.

The verification phase includes all the activities concerning the analysis and the verification of the set goals and the revision of the beliefs about the spatial relations and the present goals.

In the goal construction phase (the first phase), designer beliefs play a fundamental role. In fact they relate the formal choices (i.e. the spatial relations of the hypothetical model) of the designer with the implicit or explicit goals that the designer wants to meet in a given phase of the design process. In terms of *rational agents* the situation can be described in the following way:

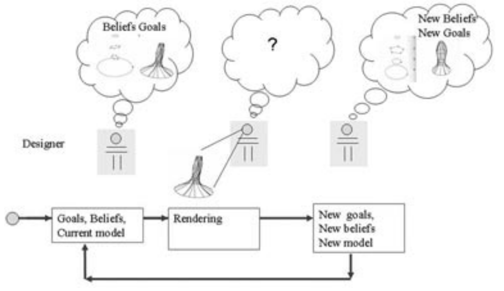


*Figure1*
*The cycle of design in a cognitive approach*

bel(designer,rx(part1,part2)$\rightarrow$ox)andgoal(designer,ox) (1)

(the designer believes that the relation rx between part1 and part2 leads to the event ox and ox is one of the designer goals).

The revision ability (the third phase) certainly constitutes an important skill of the designer, but what constitutes a kind of design competence of the designer are the beliefs rules as the ones expressed in (1), that is the capacity of the designer to foresee, to preview, to be aware, etc., that some structural choices as rx(part1, part2) carry to the desired goals (in the example the goal ox).

Believing to the implication rx(part1, part2)$\rightarrow$ ox and the subsequent observation, in the verification and revision phase, that ox is a property really existing in the generated model, are two design statements that can be related only in probabilistic terms and that are a measure of the design competence of the designer.

The planning aid system proposed in this work has been defined in such a way to provide the designer with a tool that, in each stage of the model generation process (Figure 2), makes explicit and visible the relations between the design hypothesis (functional, structural and aesthetic choices ) and the properties of the generated object. In other words, the main goal of the proposed tool is that of improving the design competence of the designer.

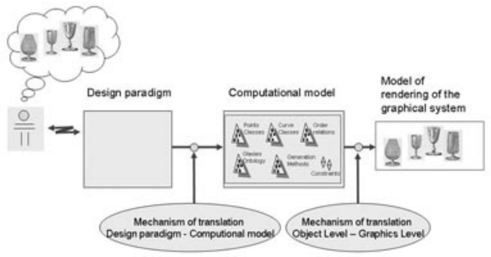

Design paradigm
Computational model
Model of rendering of the graphical system
Points Classes
Curve Classes
Order relations
Classes Ontology
Generation Methods
Constraints
Mechanism of translation Design paradigm - Computional model
Mechanism of translation Object Level – Graphics Level

*Figure 2*
*Stages of the model generation process*

To achieve our goals, we defined a specific design paradigm. This specific paradigm is still under review, but we want to remark that, every time a paradigm design is chosen, it is necessary to translate it in a rigorous computational model: in this way it is possible to construct an inferential apparatus (a program) able to control, jointly with the designer interaction, the design process.

In many research proposals of design theories (Soddu, 1998) the attention seems to be placed only on the design paradigm and not at all on on the computational model. In other research proposals, as those based on the evolutionary models, the computational model is considered but it is not separated from the design paradigm. From the conceptual point of view it is of evidence that the computational model mirrors the design paradigm, but they should not be confused: the design paradigm regards the methodological approach of the designer and its style of design, whereas the computational model concerns how the paradigm design is implemented on a computer. On the other end, from the operational point of view, a designer should be able to construct his models without to have knowledge of the adopted computational model.

In our proposal we also emphasize the separation between the computational model underlying the proposed design paradigm and the computational model of the graphic system adopted for the 3D-rendering (Maya, Rhino, LightWave, etc). We believe that such a separation is necessary because the devisers and the constructors of graphic systems place their attention only on how the shapes of the models have to be displayed and on some optimisation aspects with no attention to the architectural project in hand. In fact, analysing the most diffused graphic system it is possible to see that in the user interfaces of these systems are not considered functionalities for the management of the model generation process.

## Design paradigm: generative projects

The design paradigm proposed in this work, is composed by (a) a mechanism for the specialization of artefacts which, starting from an abstract idea of an artefact, allows the definition of an artefact more detailed and therefore more specific and (b) a mechanism for the decomposition in parts of the artefact models, where the artefact (i.e. the totality to be designed) is defined as the set of its parts and of the relations among its parts (spatial relations, constrains, etc). The specialization process (a) mimics the class generation process in a formal ontology (Guarino, 1998), that is the creation of a model B is achieved starting from a more abstract model A and introducing new descriptors for B (new features) that are added to the descriptors of A or defining constrains for B on its attributes.

According to this mechanism, the process of generation of artefacts is equivalent to the one of generation of instances of classes belonging to a well formalized hierarchy. It can be thought that every abstract model (kind of artefact or class) represents a starting point for a specific generative design. The project involves the generation of the set of instances of the considered model /class and of the set of subspecies of the model (subclasses), which may themselves generate new subspecies. In the following figure is depicted a taxonomy produced in one of the experimentations carried out.

According to the part (b) of the presented design paradigm, the artefacts are represented through:

1. the set of their parts;
2. a set of (spatial, functional, constraining, etc.) relations existing among the parts.

The approach used to represent the artifacts is explicit, indeed each artifact knows its parts. Furthermore, the capability of autonomous definitions of the parts allows use and reuse of the same parts in more than one definition of compound objects.

In this methodology there are two main artifacts' models: simple and composite artifact. A simple artifact model generation is defined by:

- the choice of a set of geometrical descriptors dj, each generated by a set of parameters pij;
- the choice of a set of relations rk representing spatial, functional, constraining relations;
- the choice of a generation method m;
- the application of the method m to the set of the geometrical descriptors dj taking in account the relations rk.

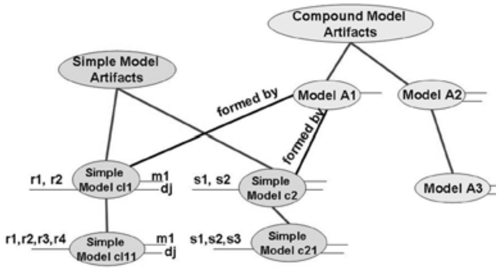


*Figure 4*
*Simple and compound models in the design paradigm*

Our design paradigm includes constructs (or objects of the design interface) able to define simple artifacts' models. For instance we can define set of simple models by varying the geometrical descriptors dj, df, ..., the methods m1, m2, ..., the relations rj, sk, ....

The generation of a compound artifact model T is defined by:

- the choice of a set of (simple or compound) components cx(T) belonging to the artifact T;
- the choice of a set of connection relations among the parts;

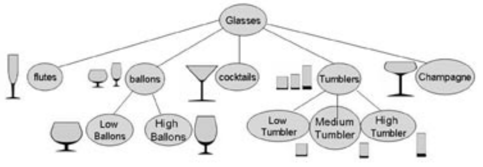


*Figure 3*
*An example of artifact taxonomy*

- the choice of a set of constraints represented by the relations vt among the parameters pij of the components ci of the compound object.

Also for the compound artifacts our design paradigm includes constructs (or objects of design interface) to define classes able to generate the relative models.

The proposed design paradigm gives the designer the chance to define a generative project through the definition of a (super) class representing a compound artifact model represented as a whole composed by parts. From this model/class is possible to generate a substantial set of artifacts' instances used by the designer to control the generation process and to lead it to desired design goals.

The break down process (b) is iterative according to the schema depicted in Figure 5.

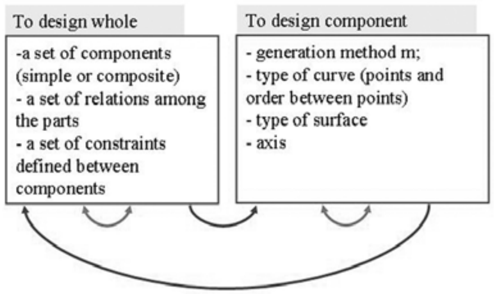

To design whole
-a set of components (simple or composite)
- a set of relations among the parts
- a set of constraints defined between components
To design component
- generation method m;
- type of curve (points and order between points)
- type of surface
- axis

*Figure 5*
*The iterative cycle of whole and components generation*

## The computational model

In the methodology we propose the design paradigm is independent of  the related computational model, even if the latter can encompass some specifics reflecting paradigm features. Actually a class of a frame-based ontology is created, corresponding to a reference model used by the designer in the paradigm. During the specialization process of a model-artifact, driven by the designer, that class will be the superclass common to all the subclasses successively defined.

From a computational point of view in the model are represented all the entities necessary to generate an artifact model. At this level there are domain independent parts (basic classes of every model that can be generated) and parts dependent of it (the particular artifact model). The latter parts are builts upon the independent parts. They are tied to the particular artifact model and are generated by the interaction of the designer with the design interface.

### Built-in Classes

The parts of the computational model that are independent of the domain (artefact) are composed by some built-in classes of a frame based ontology: Points, Axes, Order Relations, Curves (primitives definable as Free Form for ordered points), Generation Techniques. In the figure are depicted the main model classes independent of the domain.

Each class of the domain is defined by means of

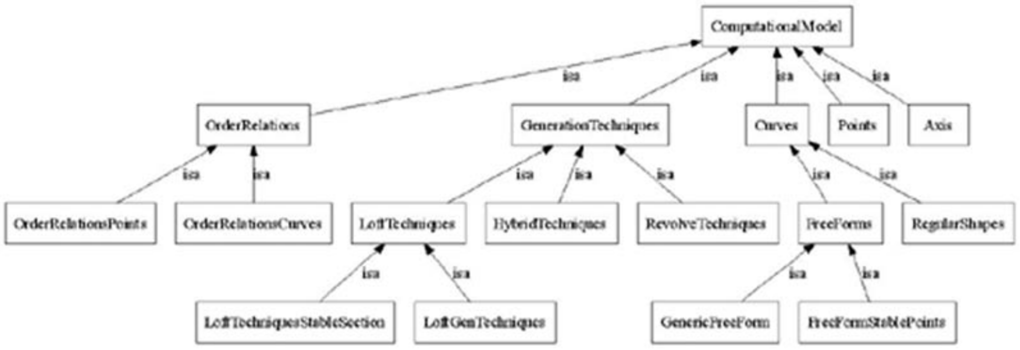


*Figure 6*
*Built-in classes of the computational model*

specific attributes (see the definition of the Generic FreeForm class in fig. Xxx and its attributes CurveName, CurveOrdPoints, CurveSetPoints and respective types):

Curves::FreeForms, FreeForms::GenericFreeForm.
GenericFreeForm[
    CurveName=>string,
    CurveOrdPoints=>>OrderRelationPoints,
    CurveSetPoint=>> Points]

(The class of the generic curves GenericFreeForm is a subclass of the curves FreeForms that is in turn subclass of the class Curves.

The class GenericFreeForm has the following attributes: CurveName which is of type string, CurveOrdPoints which is of type OrderRelationPoints and CurveSetPoint which is of type Points).

## The classes of the model and the Part-Whole relations

In correspondence with the models defined by the designer, in the design interface will be built some classes that represent the knowledge about the generation of the models. As an example, in Figure 7 is depicted a class that can be used in a generative design for the generation of a model of a glass. This model can be generated both using a single generating technique for all the glasses and using mixed technique allowing the parts (cup and stem) to use different techniques (loft and revolve).

In a generative design, where the designer breaks down the artefact model in parts, the system we realized constructs explicit not taxonomic relations (i.e. Part-Whole relations) among the model classes that represent the totality and the model classes that represent the components.

## Constraints associated to the classes

In the generation phase, the designer identifies the parts to create and for each of them he chooses the geometric shapes and the generation methods that will be used in the production of the model.

In the model definition, among the independent parts there are the geometric entities: axes, points and curves. They represent the basic elements for

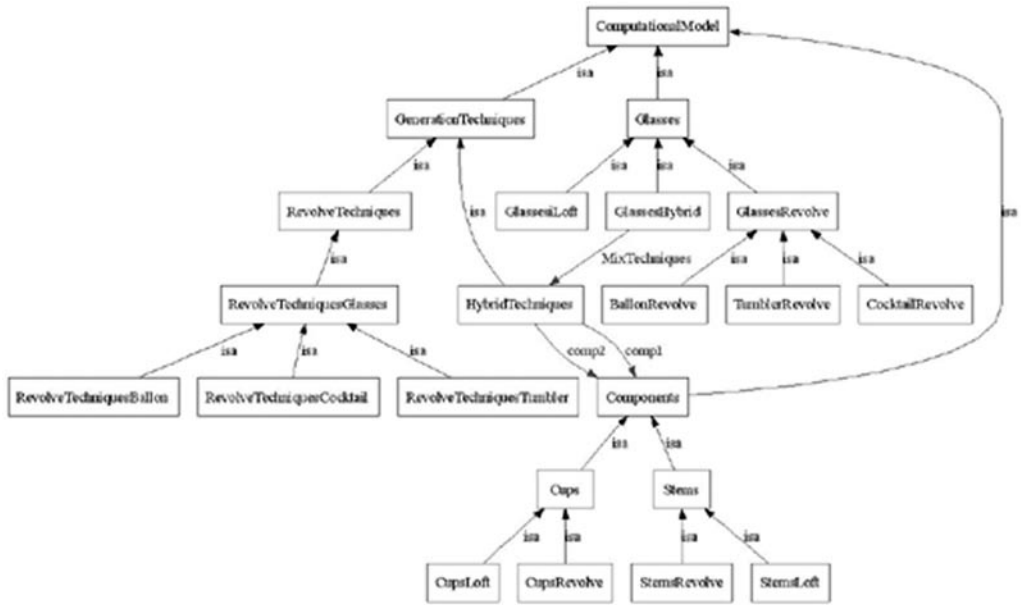


*Figure 7*
*Example of a computational model of a specific domain*

the generation of the whole model or of one of its components.

In the definition of the geometrical forms, the designer may choose, for instance, some control points constrained to some value ranging in limited areas. In this way he will have a greater control on the creation of the final model. Considering these features of the design paradigm, the system we realised allows the construction of subclasses starting from the basic class and allows the association of constrains to these subclasses. Using this mechanism, the designer creates more and more specific model classes adding new models (subclasses) and defining new constraints for each model. In conjunction with these design choices the presented system creates new classes along with the associated constraints.

In the present version of the achieved system, the adopted constraints language is PAL (Protegé Project). The following code fragment shows an example of a constraint associated to the class basicPoints which is a subclass of the class Points.

```
basicPoints::points.
basicPoints[letteral=>string, coordX => floit, coordY => floit, coordZ=> floit]

(forall ?x (forall ?y (forall ?z
        (and(> (coordX ?x) 0.0)(<(coordX ?x) 65.0)
                (<(coordY ?y) 35.0)(>(coordY ?y)0.0)(=(coordZ ?z) 0.0)))))
(defrange ?x :FRAME basicPoints)
(defrange ?y :FRAME basicPoints)
(defrange ?z :FRAME basicPoints)
```

In the above example:

- the class basicPoints is a subclass of the class points (basicPoints::points);
- the attributes of the class basicPoints are coordX, coordY and coordY whose type is float (floating point);
- the coordinates of the class basicPoints are constrained for the coordinate X in the range 0-65 and for the coordinate Y in the range 0-35, moreover, there exists the constraint to lie on the plan (the coordinate Z is set to 0)

Every constraint of a given class A is propagated (without ulterior specifications) to all the subclasses of that class.

## Correspondence between the computational model and the graphical system

The graphical level is the final part of the generative process. It produces the artefacts generated by the model that has been defined by the designer at the design level.

This process phase allows the representation of the model that the user wants to realize at the object level. Here are created the concrete objects (the instances of the model classes) that will be displayed in the chosen system.

The correspondence between the computational model of the adopted methodology and the chosen graphical system allows the rendering of these objects. Due to such a correspondence all the models defined by the designer by means of the user interface and translated into the computational model will be displayed by the chosen graphical system without the necessity to produce additional code for the visualization.

## The system design interface

In this section some screen shots of the interface of the implemented system are presented. This interface concerns with the design paradigm. We want to remark that among the main design metaphors we considered: (1) the class/subclass specialization (openly inspired to the formal ontology modelling scheme (Guarino 1998)), (2)

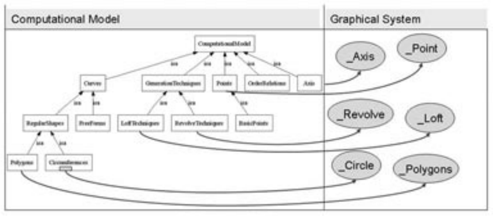
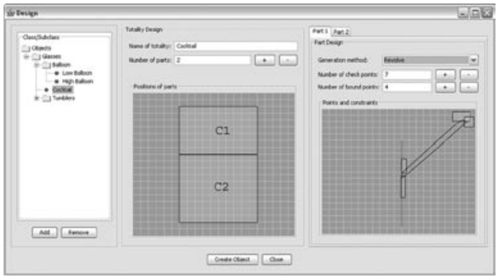
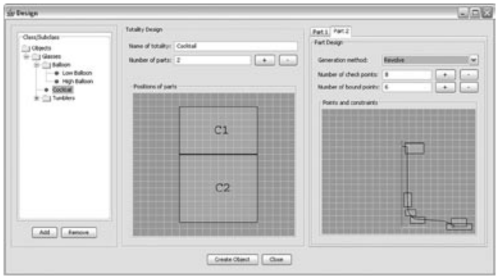


*Figure 8*
*Correspondence between the computational model and the graphical system*



Name of totality: Cocktail
Number of parts: 2
Generation method: Revolve
Number of check points: 7
Number of bound points: 4

*Figure 9*
*Screen shot of the design interface: cup design*



Name of totality: Cocktail
Number of parts: 2
Generation method: Revolve
Number of check points: 8
Number of bound points: 6

*Figure 10*
*Screen shot of the design interface: stem design*

the decomposition of a totality in parts (inspired to the spatial reasoning representation theories (Calabrese et al 2005; Artale et al 1996) and (3) the possibility to define constrains associated to the models (also straight inspired to the formal ontology theory).

We want also emphasize that even if the presented interface is directly inspired to the presented computational model, the design paradigm is completely distinct. In this respect see the diagram depicted in Figure 4 which represents the specialization mechanism with the real diagrams of the computational model in Figure 7. The first diagram represents a non formalised methodology whereas the others show a representation defined through a specific formal language.

The system interface allows the design of generative projects according to the proposed methodology by means of the design metaphors (1), (2) and (3).

It allows to define:

- classes of models and their respective subclasses;
- models, represented as the set of their constituent parts and of the constraints existing among them,
- single parts, their graphical generation model and their control points and the existing constraints.

## References

Artale A., Franconi E., Guarino N., Pazzi L.: 1996, Part-Whole Relations in Object-Centered Systems: An overview, Data & Knowledge Engineering (DKE) journal 20 pp. 347-383 – North-Holland, Elsevier, 1996

Calabrese A., Licenziato L., Mele F., Minei G., SorgenteA., TalamoO.: 2005, Spatial reasoning for 3D visualization and reconstruction of architectural artifacts, Convegno AI*IA Milano 19-Ott-2005

Coppola C., Calabrese A., Iazzetta A., Mele F., Talamo O.: 2005, The transformation's control and development, ECAADE 2005, 22nd International Conference Education and Research in Computer Aided Architectural Design in Europe September 21-24, 2005 Lisbon, Portugal

Flora2 Project, http://xsb.sourceforge.net/Guarino N.: 1998, Formal Ontology in Information Systems, Proceedings of FOIS'98, Trento, Italy, Amsterdam, IOS Press, 6-8 June 1998

Pontecorvo M.S.: 1999, Designing the Undesigned: Emergence as tool for design, in Generative Art, Dedalo, Roma 1999

ProtégéProject, http://stanford.protege.eduSoddu C.: 1998, Generative art, Dedalo, Roma 1998

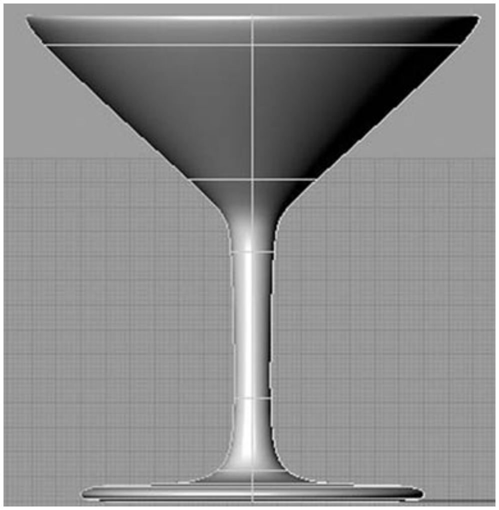


*Figure 11*
*Screen shot of the design interface: rendering of the glass*