INTRODUCTION

Post-modernism has brought the principle of uncertainty to our everyday lives. How can an individual from a very backward society like one from Taliban Regime be suddenly questioning the whole of the so-called Western civilized world? It is obviously beyond the scope of the Scientific Report to delve deeper into this issue but being skilled in matters of energy conservation and being sure all men are created equal, this does not puzzle me. To the contrary this enhances my faith in God and makes me relaxed: someday sustainable cities will be a reality on Mother Earth![1]

Not being able to express the truth as art, science and religion have been promising since Adam and Eve first appeared on the surface of Mother Earth, the ordinary man (again at least the most evolved ones) are realizing the importance of betting on mistakes. A far cry

[1] Thoughts in the Presence of Fear
by Wendell Berry

I. The time will soon come when we will not be able to remember the horrors of September 11 without remembering also the unquestioning technological and economic optimism that ended on that day.

II. This optimism rested on the proposition that we were living in a "new world order" and a "new economy" that would "grow" on and on, bringing a prosperity of which every new increment would be "unprecedented."

III. The dominant politicians, corporate officers, and investors who believed this proposition did not acknowledge that the prosperity was limited to a tiny percent of the world's people, and to an ever smaller number of people even in the United States; that it was founded upon the oppressive labor of poor people all over the world; and that its ecological costs increasingly threatened all life, including the lives of the supposedly prosperous.

IV. The "developed" nations had given to the "free market" the status of a god, and were sacrificing to it their farmers, farmlands, and communities, their forests, wetlands, and prairies, their ecosystems and watersheds. They had accepted universal pollution and global warming as normal costs of doing business.

V. There was, as a consequence, a growing worldwide effort on behalf of economic decentralization, economic justice, and ecological responsibility. We must recognize that the events of September 11 make this effort more necessary than ever. We citizens of the industrial countries must continue the labor of self-criticism and self-correction. We must recognize our mistakes.

from Modern Times, Post-modern Times demands courage and courage to bet on insights and the magic word called experience.

Likewise the current state of uncertainty seems to start pervading the field of computer science after software systems based on specification keep failing to mirror the real world.

Computer scientists strive to cope with software evolution. The most traditional software developers to the most daring one are being possessed by its spirit. Insofar as it has become the heart of the approach called *Extreme Programming **XP*** encompassing indeed the so-called **agile methodologies** to contrast to the previous ones **heavy methodologies** based mainly on specification.

In the programming world, "extreme" describes a methodology in which teams collaborate on projects that entail risk, explore uncharted territories, setbacks and failures provide essential feedback on which the software development process thrives, where risk is something to be understood and managed not merely avoided.

So **courage** is a fundamental element in X-programming. Socrates had already perceived that *courage* is the Queen of the virtues. All virtues are useless without courage! So essential elements of X-programming are:

- **Simplicity:** Simple things are easy to create, maintain and understand.
- **Communication:** Software development is a cooperative game of invention and communication
- **Learning:** XP concentrates on building an evolving, shared vision of the project among the customer, the development staff and related application domain interests as essential to the success of any project.

Alistair Cockburn from Crystal Methodologies goes further and characterizes people as non-linear, first-order components in software development![Cock01].

I am in agree with him and obviously this explains why I have been managing to unfold a daring approach to solve the main problem of mankind: the need for shelter. Some would object, food is the most important one. Well sustainable cities integrate rural and urban areas, hence tackle the problem of food directly. Still others would say, education is the most important problem of mankind. Finally I am in agree and this is why I want to become a movie maker: to help mankind to be courageous.

To ease this last view and promote the cause of biodiversity, I described below briefly the essential aspects of my childhood and adolescence that made me challenge conventional standards of education and bet on things that I believed would make a difference to create a better world. The price to do this continues to be high but being supported by the enveloping metaphysical laws enhances my persistence. Although sometimes events like the energy crisis in Brazil that I thought would happen only around 2030 scare me. But above all I see God's love towards mankind in these gestures, sendings signs to mankind to avoid a final disaster in time.

Having been born in an illiterate family and having to discover the world by myself with my own eyes, I got involved with Biological Sciences, especially Genetics and Ecology as early as twelve years old. At the age of sixteen, I was already disappointed with rigid scientific laws that were unable to alleviate my great existential crisis.And suddenly an artistic vein emerged and nothing in the world seemed to me more precious than to follow urbanism! Yes, to create models by computer to transform the world!

Miraculously the need to know how to draw was abandoned as a priority to follow Architecture. So I had no problem to be admitted to an architectural school in Porto Alegre!

What a disappointment! Obviously I could not swallow things that were impinged on me such as Greek standards of art and architecture! I abandoned it and restarted my biological search advised by my ex-husband, a researcher in computer science strongly disappointed with its status quo, already aware it would lead nowhere!

After an attempt of suicide, I came back to architecture when I was 22 years old. This time I was thoroughly possed by art and the passion for architecture. Again in Rio de Janeiro, miraculously I did not need to know how to draw! But the same Greek standards of beauty continued to be the mainstream concern of the school! Needless to say that chaos installed itself in my life.

Well I managed to survive and a myriad of metaphysical laws keep revealing to me every day. And life on Earth is becoming more and more meaningful and palatable! The more my professors in Rio de Janeiro awarded me zero grades for research evaluated as of a master's course but that was not demanded and would provoke imitation by the anarchical cariocas the more I got happy because I discovered I was compensated in time to advance my studies as I needed. I mean the era of strikes began, then a chaos was generated and I was easily compensated by the generous tens distributed to everybody for doing nothing due to the strikes. Of course the students enjoyed themselves and simply started not to attend classes, forcing the total change of old-fashioned ways of teaching based on tests that were not even told when they would happen in advance!

So due to metaphysical laws simple as "nothing resists an effort of concentration", life became enjoyable and brought me the discoveries of many more complex ones.

So I view this minimalist approach in X-programming with hope. First because it thrives on the way experience is and above all the way all scientific reasoning starts. The need for becoming empty and start filtering what really matters is sound!

The perception that several heads reasoning is better than one is still far better! Indeed this tunes with the semiotician Charles Saunders Peirce's characteristic definition of the real:

And what do we mean by the real? It is a conception which we must first have had when we discovered that there was an unreal, an illusion; that is, when we first corrected ourselves. Now the distinction for which alone this fact logically called, was between an ***ens*** *relative to private inward determinations, to the negations belonging to idiosyncrasy, and an* ***ens*** *such as would stand in the long run. The real, then, is that which, sooner or later, information and reasoning would finally result in, an which is therefore independent of the vagaries of me and you* [Apel81:28].

This effort visible in his 1868 essays represents a concretization of the idea of "consciousness in general" in the direction of a "postulate of practical reason", in Kant's sense. Thus Peirce adds the following comment to his definition of reality: *Thus, the very origin of the conception of reality [namely, from the difference between my idiosyncrasy and that which proves to hold as an opinion "in the long run"] shows that this conception essentially involves the notion of a* ***COMMUNITY,*** *without definite limits, and capable of a definite increase of knowledge* [Apel81:28].

Moreover this stuff is important because it lies in the roots of the trend the actual sciences embraced and the cause of the divorce from phenomenology or the description of the real thing or phenomenon (in computer science it is responsible for concern with specification rather than with evolution[2]). Hence the misunderstanding of the nature of the sensitive that is pejoratively assumed to be as unable of true intelligibility. And of course the inability to

[2] See I.3 .Specification versus Evolution and I.4 S-types versus E-types

reveal the morphodynamic level [3] underlying the phenomena. Apel expresses this beautifully: *the meaningless presupposition in the modern theory of knowledge lies, according to Peirce, in the implicit assumption at work in Ockham, Descartes, Locke, and Kant that cognition is blocked off from the things actually to be known by its own causal mechanism and so has primarily to do with the effects of things in the receptaculum of consciousness, while the external things remain incognizable as "things-in themselves".* Peirce opposes himself to this view. He criticizes the meaning, Here lies his original thought. Peirce interpreted Kant's "transcendental synthesis of apperception" as a "reduction of the manifold of sense data to the unity of consistency" by a hypothesis. The essence of knowledge lies for Peirce in the formation of opinion (representation, belief) through conscious or unconscious inference.His semiotic transformation of the concept of knowledge implies primarily in a unified, perceptually schematizable representation (Vorstellung) of the world in contrast to Kant's that identified the essence of knowledge in the formation of a semantically consistent opinion, consisting this in the ability to imagine things in a spatial, temporal world of appearances.

This is also the so-called hermeneutic way of being. The scientific research dedicated to the science of art is aware that one can neither replace nor supplant art experience. The phenomenological return to aesthetic experience teaches that this ponders over what it experiences, uncovering the genuine truth. In thinking, the only thing that makes emerge what is in the thing, is the thing itself that manages to reveal itself while we are engrossed

[3] Jim Coplien in joint research with Liping Zhao is trying to formalize the notion of expressive powers of multi-paradigm design using models from symmetry theory to capture the expressive power of language features. He intends to go beyond symmetry theory into symmetry-breaking as a foundation for patterns. Symmetry is about commonality invariance; symmetry-breaking is about variations [CZ00].

in the thing.[4] This happens like a game. And the only prerequisite is to know how to play: I mean to abandon oneself to the force of thinking. The immanent need of the reasoning reveals the next step, inhibiting any preconceived idea that can occur to oneself while the new conceptions are shaping and striving to bloom. Hence the thing reveals itself as a successive multitude of inner and outer parts.

When **agile software developers** suggest the **incremental and iterative strategies**, they are recognizing the search for truth is hard and a trial and error process. To prefer to fail conservatively than to risk succeeding differently as Alistair Cockburn puts forward explains the persistence of **heavy methodologies**. Having felt chaos inside me makes me realize this is not an elementary cognitive process to tackle. However without being able to void oneself, the ability to identify oneself with the nature of the thing becomes difficult. I felt this all the time in the outstanding discussion we held about object oriented architectural evolution [MG01] at the Workshop in the last ECOOP'01 in Budapest last June. By two o'clock after five hours of exercise of absolute freedom to express our thoughts, a beheading occurred. The right thing to do then would be to go home and let the higher ideas visit this chaos we were feeling and little by little organize it shaping new conceptions. We had to continue the discussion and suddenly reactionary views installed. When we wrote the final report, the need for a metaphora based on architectural practice (building architecture) was necessary to not highlight the emergent contradictions between very soft views and more conventional ones based on layering architecture. When chaos visits you, the trend is to let new ideas flow. If one blocks this flow, the contradiction becomes blatant!

[4] I analyzed this process in [Lour00]. It is equivalent to forget about one's ego and to transfer one's consciousness to the being ot the thing.

So the real bottleneck is the human cognitive processes themselves. Hence the focus on people as first-order components in software development as I emphasized in [Lour01] and [Lour00] is not an exaggeration.

When we wrote the Workshop Report a rather iterative strategy was adopted. We reworked the pieces of the system. It is learning by completing. Booch gives this sort of learning a value by calling it "gestalt, round-trip design". It is hard to plan because it is hard to guess in advance how many major learnings will take place. Then I suggested we should discuss things by e-mail again. Nobody even me had more time to do so. If this had been accepted, we would be engaged in incremental strategy. Hence different pieces of the report developed at different rates or times, and integrated as they are developed would emerge. It addresses new learnings in the team's overall way of working. After a section of the report was built, the team would examine their working conventions to find what should be improved. But this means to not report exactly what was discussed at the Workshop! And Galal simply saw no problem in doing this next ECOOP'02 in Spain! Although Alistair Cockburn sees the last one as easier to implement, it didn't work in the context of different people in different cities trying to write a Workshop Report. So the overall report did not stick to the pure hermeneutic way it had started! However I found this experience superior to the traditional Workshop held in Vienna [Lour01] despite the mature researchers that were there and the valuable discussions I have been pursuing with some through e-mail!

I would conclude that if researchers do not manage to unfold semiotic cognitive processes at the level of discussion, they would never build software systems that introject these processes.

Hence the current computer science practice is trying to unfold semiotic systems but still it is a far cry from the genuine semiotic human minds. My knowledge system is an effort in this direction. Reading further the reader will perceive the difficulties it faces.

To make computer scientists more aware that it is possible to build semiotic systems I will briefly try to summarize what they mean.

The concept of symbolic machine has become a common metaphorical designation of the computer. Computer semioticians have reasons to generalize this to semiotic machine: able to create processes of sign production and interpretation (i.e., processes of *semiosis*).

Peirce defines semiotics as the *the doctrine of the essential nature and fundamental varieties of possible semiosis*, and semiosis as *the intelligent or triadic action of a sign which involves a cooperation of three subjects, such as a sign, its object and its interpretant.* [Nöth02]. Peirce also assumes *the interpretant is ... a sufficiently close analogue of a modification of consciousnes* [Nöth02].

Winfried Nöth describes the semiotic field from less to more complex semiotic systems as a gradual continuum from less complex to more complex processes of sign processing. Among the less complex processes are those merely mediated by instruments or technical devices such as a thermometer, a thermostat or the system of an automatic traffic light usually dealt with as S-types (specification types). The most complex processes of semiosis occur in living systems. Hence semiosis is not restricted to sign production and interpretation in humans. There is semiosis in matter, machines, biological systems and human minds.

Moreover processes in which machines serve as mediators in human semiosis are certainly processes of genuine semiosis.If a traffic sign serves as a genuine sign to a driver, an automatic traffic light is no less a genuine sign. In this sense, sign processing in the

interface between humans and computers is genuine semiosis. Signs are produced by humans, mediated by machines and interpreted by humans. In such processes of computer-mediated communication, the computer serves as a semiotic extension of human semiosis.

A tricky situation develops when we think about the fact that the sign cannot be localized in the brain alone, but must also be sought in the signs that result from the brain activity. Peirce solves this: *it is much more true that the thoughts of a living writer are in any printed copy of his book than that they are in his brain* [Nöth02]. Peirce also expresses the idea of the unity of sign and thought as follows: *It is wrong to say that a good language is important to good thought, merely; for it is the essence of it*[Nöth02].

Moreover Nöth emphasizes that writing and written calculus are not mere semiotic alternatives to speaking and mental calculus, but operations which permit the development of more difficult arguments and the solution of more difficult problems since the fixation of the signs on paper have the advantage of increasing our memory. This effect of the externalization of our memory is one of the reasons why thoughts come to a writer while writing on paper. The thoughts that come to us when we speak are not the same ones as those that we express when writing on the same subject. So there is quasi-mind not only in the brain but also in the machine.

Another genuine feature of semiosis that Peirce used to define is self-control. In systems theory the term autopoiesis is used to describe a system which evinces this kind of autonomy due to self-control. When the control comes from elsewhere, from outside, the system is an allopoietic system.

Today the distinction between allopoietic and autopoietic systems and more generally between engineering and biology is no longer as clear as it always seemed to be. On the one hand, doubts concerning the genuine autonomy of human consciousness have been raised.

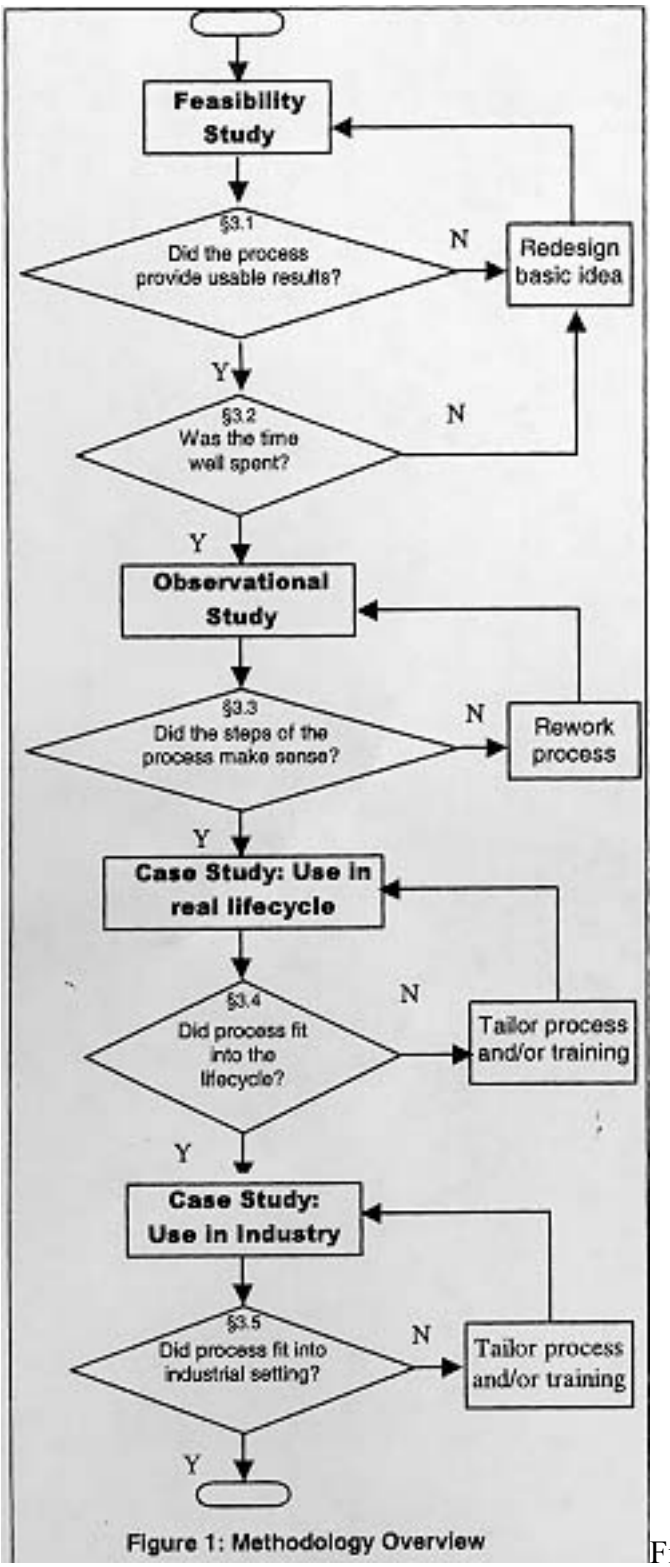
Free will is hard to achieve and one must be beyond genetic and cultural factors that determine behaviour. These evidences from evolutionary biology and contemporary genetics show the autonomy of human action and the destiny of humans are determined by factors independent of the self.

On the other hand, we are being confronted with the development of computer programs, automata and robots which no longer seem to be mere allopoietic artifacts but begin to evince features of autopoietic systems.

The recursive reflective abilities displayed by my ecodesign model and the wise interaction between man and machine and even the cooperation among humans mediated by the computer mirrors a true semiosis.

Moreover the fact I am aware that the model has goals in itself as defined by the concept of entelecheia from Aristotle makes it indeed like a living organism. And of course this may be true of any other system/machine that is mediating the progress of mankind.

To date there is a growing interest in empirical study in software engineering for validating mature technologies and for guiding improvements of less-mature technologies. Shull et al [SCT01] introduce a methodology based on experiences for taking a newly proposed improvement to development processes from the conceptual phase through transfer to industry. The methodology presents a series of questions that should be addressed to inspection processes for object-oriented designs [Figure 1].


Feasibility Study
§3.1
Did the process provide usable results?
N
Redesign basic idea
Y
§3.2
Was the time well spent?
N
Y
Observational Study
§3.3
Did the steps of the process make sense?
N
Rework process
Y
Case Study: Use in real lifecycle
§3.4
Did process fit into the lifecycle?
N
Tailor process and/or training
Y
Case Study: Use in Industry
§3.5
Did process fit into industrial setting?
N
Tailor process and/or training
Y
Figure 1: Methodology Overview
Figura 1.

PART I

I. MULTI-PARADIGM DESIGN EASES KNOWLEDGE SYSTEM EVOLUTION

Before advancing my inter-, multi- and transdisciplinary research concerned with its computational aspects, it is of the utmost importance to sharpen terminology. However this is not within the scope of the Scientific Report.[5]

It is of the utmost importance to focus here on the current questioning of "Software Engineering" as a wrong metaphor on one hand.

On the other hand, one cannot understand the title of this chapter without pondering over what paradigm means.

In the following sections of the first part I will try to introduce approaches that enhance the need for a review of the current paradigms in computer science especially those concerned with software evolution.

Software developers that have followed the conventional trend so well explained by Michael Jackson are reacting and trying to adapt their software theories to the need of

[5] I will try simply to be clearer and try to avoid the current widespread use of words such as layers, components, views without any relation to their granularity. Indeed terms such as software evolution, software architecture, software engineering have different meanings depending on the context they appear.

simulating the real world and its inevitable dynamic nature translated as changes, evolution, adaptation and so on.

In the second part I introduce the phenomenological concern with the nature of the things and the sophisticated conceptualization in terms of semiotics, hermeneutics and autopoiesis in a rather synthetic way.[6]

Moreover I try to put forward briefly how the so-called agile software developers are trying to run away from and replace the so-called heavy methodologies that are described rather as S-types independent of the adopted paradigm.

Finally in the third part I introduce my version of the multi-paradigm design inspired by Jim Coplien's PHD thesis [Copl00], however within the context of prototype based object oriented programming languages such as Self designed by Randall Smith and David Ungar. Sensitive to the criticism of the tyranny of the object, they created a subjective version of Self called Us [SU96].[7]

I.1 The questioning of software engineering as a metaphor

In recent keynotes, Michael Jackson's *Where, Exactly, is Software Engineering?* (Joint 8th European Software Engineering Conference (**ESEC**) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering **(FSE-9**) held in Vienna September 10-

[6] A more encompassing way was introduced in the First Scientific Report entitled "O desdobramento e o modelamento da consciência artística na Infoera. Outubro de 2000.

[7] Due to the short time of a year to review extensively what is going on in computer science and delineate an up-to-date research plan still consistent with the philosophical theme of the Model of Primary, Secondary and Tertiary Waves to generate sustainable cities, I made up my mind to keep the third part in this chapter. Of course it is a consequence of previous concepts that were exposed in this chapter.

14/2001), James Owen Coplien's *It's Time to Kill Software Engineering* (XV SBES - Simpósio Brasileiro de Engenharia de Software; Primeira Conferência Latino-americana em Linguagens de Padrões para Programação e XVI SBBD- Simpósio Brasileiro de Banco de Dados –October 1-5/2001, Rio de Janeiro) and Jim Highsmith's *Is "Software Engineering" the Wrong Metaphor? And Why Should We Care?* (OOPSLA 2001- ACM Conference on Object Oriented Programming Systems, Languages and Applications, October 14-18, 2001, Tampa Florida) are exploring the positive and negative roles the "software engineering metaphor" plays in various types of software development efforts and discusses the practicalities of other metaphors as a cure to the problems we face. Jim Coplien toys with other metaphors such as Medicine, Art, Cockburn's game theory, Software as Literature (it is how great programmers view it- mediocre programmers view it as engineering; beauty matters, a creative act of pure thought stuff, done in community, "group poetry writing", Gabriel's vision of the Erotic Life of Code [Gabr01]) and finally puts forward what it means to have a Master of Fine Arts in Programming:

- Code is the material
- We read code
- We write code
- We compile, test and deliver code
- Customers pay for code- and documentation
- An MFA in literature would not deliver outlines of story boards either for degree of profession
- How many software engineering professors- or practitioners – have written code recently?

He challenges the audience: *A small group of us would love to work with your institution to define what it means to have a Master of Fine Arts in Programming. If you are interested, please write me at JOCoplien@ cs.com.*

And concludes citing Jones, 1988: *The more I see of software designing the more I notice resemblance not to design in other fields but to craftsmanship. In each the designing, if such it can be called, is done by the maker, and there is much fitting, adjusting, adapting of existing designs, and much collaboration, with little chance of a bird's eye view, such as the drawing board affords, of how the whole thing is organized, though, in craft evolution, if not in software, the results have the appearance of natural organisms or of exceptionally well integrated designs. But there is an important difference: software is increasingly made by modifying the actual material of previous pieces of software, as a building may be altered for a new use, whereas a wagon-maker, for instance, modifies the form, but does not reuse the material, in making each small step in the gradual evolution of his product. As in natural evolution, each alteration is made without conscious intention, or plan, of what kind of artefact may later appear out of the seemingly blind process of making corrections, here and there, as and when lack of adaptation to the working conditions, or to the materials or the making process, becomes evident. But there is a tremendous respect for the form, as it has evolved so far, embodying, as it does, the otherwise unrecorded history of a thousand ways in which the artefact and its context can be attuned.*

Martin Fowler, one of the seventeen software developers of the Agile Alliance [Fowl01] is also unhappy with the usual inspiration for methodologies based on engineering disciplines such as civil or mechanical engineering. Such disciplines put on a lot of emphasis on planning before you build, such engineers will work on a series of drawings that precisely

indicate what needs to be built and how these things need to be put together. He continues his argument and concludes:

- In software: construction is so cheap as to be free
- In software all the effort is design, and thus requires creative and talented people
- Creative processes are not easily planned, and so predictability may well be an impossible target
- We should be very wary of the traditional engineering metaphor for building software. It's a different kind of activity and requires a different process.

In the discussion my supervisor and I joined in the *IVth Workshop on Object Oriented Architecture Evolution* held in conjunction with the 15th European Conference on Object Oriented Programming (ECOOP'01) [MG01] and my recent paper presented at the *IVth International Workshop on Principles of Software Evolution,* Vienna, September 10-14/2001 [Lour01] entitled *An evolutive architecture reasons as a semiotic, hermeneutic and autopoietic entity* , software was likened to art, especially jazz Moreover I put forward that a knowledge system (domain level + architectural level + code level) can be an agent in genuinely semiotic processes.

Indeed Winfried Nöth in Semiotic Machines [Nöth02] argues that the distinction between engineering and biology and allopoietic and autopoietic systems is no longer clear. In my master's dissertation [Lour88], PHD thesis [Lour98] and my last Scientific Report to Fapesp [Lour00] I tried to blur this distinction especially supported by recent arguments put forward by post-quantum Physics [Sarf00]. Metaphysically speaking I am thoroughly aware of the "alive nature" of my *Model of Primary, Secondary and Tertiary Waves to generate sustainable cities*. To illuminate this idea, I introduce the notion of *entelecheia*

from Aristotle [Arist56:26]. The meaning of *entelecheia* is literally to have an aim (*telos*) in itself or have its determination in itself. This is the own definition of life or of an organism. However what I tried to put forward is a transdisciplinary notion. In a scientific community that has problems to understand what interdisciplinary means, even worse unable to assimilate the idea of multidisciplinary (to work on the frontiers of many disciplines where a fusion must emerge and hence something different from each discipline), to understand what transdisciplinary means is to ask too much. However transdisciplinarity has such power and mimics even the power of life, rather spirit whose stuff like thought is transmitted at a speed higher than light speed that I do not worry very much with these misunderstandings.

Hence it is my goal to await patiently the gradual but firm evolution of mankind towards more encompassing semiotic views. It seems lots of progress is being made within the context of computer science towards it in the last two years. Indeed in my last Scientific Report for FAPESP I argued for the perception of unity of art, science and religion as a sound basis for the emergent science of consciousness. So my criticism of software engineering principles fathoms invisile realities that shape reality. I cannot speak out this metaphysics basis blatantly because it deals with cognitive processes connected to the soul. Of course the average human being is not aware of them or has not awakened them.

I.2 A practical definition of paradigm in the context of a programming language

Multi-paradigm design tries to dig deeper than any single technology or technique to address the fundamental questions of software abstraction and design. What is a paradigm? asks Jim Coplien in his recent PhD. Thesis [Copl00]

What is the relationship between domain modeling, software architecture design and implementation? These questions go to the foundations of abstraction and conceptualization that underlie the basic paradigms of domain modeling, computational design and implementation through programming languages. Coplien shows the greatest enthusiasm appealing to the role of broad insight and experience rather than to rely on the first principles that are the hallmark of most academic research. He criticizes current research in computer science based on a new-found nugget in the discipline and which expend much energy to evaluate and validate such an idea in a limited context.

His thesis opens the gate to an encompassing exercise of software in the direction that I have been putting forward [Lour98]. Curiously recently knowledge system community is realizing that a knowledge system is also a sort of software system and can thrive on object oriented development.

. A knowledge system is in fact a software system [Schr00]. In that respect, a software development methodology may be applied in developing a knowledge system. Both knowledge systems and software systems seem to be merging in the sense of recognizing that it is possible to identify and model the necessary domain knowledge for supporting object-oriented framework development.

However what might be the obstacles that one experiences in mapping domain knowledge into object-oriented frameworks?[AMT00] In the current literature of both artificial intelligence and object-oriented software systems, a sensation of a Babel effect misleads even fundamental mainstream endeavours that have a great potential to change dramatically the quality of life on Mother Earth.

Why is this Babel effect there? In the next section it will be shown the root of the problem that started with the mainstream approach adopted to develop software known as specification. It pervades all paradigms in computer science. The gap between software systems and real world systems has motivated a strong concern with the need for evolution.

On the one hand, research on software evolution focuses on empirical investigations studying changes in long-living software systems and on methods and tools how such evolutionary behaviour of software can be controlled or supported.

On the other hand, the multiplication of paradigms to tackle this instead of being a solution is rather leading computer science to stagnation.

The field of research called semiotics should throw light in this chaos. However it is a strongly hermetic field. Semioticians in general are unable to build a bridge to the application fields and semiotics remains as an "unexploited mine in the middle of the Amazon forest".[8]

Dirk Bäumer [Bäum00] tries to introduce UmbertoEco's semiotics to the software development in his PHD thesis. This led him to perceive that there should be homologous reasoning structures in all levels of the software system, from domain level passing through the architectural level and to the implementation level. His thesis becomes consistent, a far

[8] Other important source for modeling is catastrophe theory introduced by René Thom. The difficulty with these methodologies is that they require an extensive and deep knowledge of the field to be modeled.

cry from most work in computer science, however since he programs in C++ and basically orbit around class-based constructions, he gets into trouble to unfold a seamless thought.

Similar difficulties met by computer experts `have provoked the identification of the problem as *the tyranny of the dominant decomposition* [OT99]:modern languages and methodologies permit the separation and encapsulation of only one kind of concern at a time. Examples of tyrant decompositions are classes (in object-oriented languages), functions (in functional languages), and rules (in rule-based systems). However the vast majority of the programming community still hopes to find the solution in

better programming languages or approaches. They try to avoid to go to higher levels such as the design and domain levels. Especially the domain level is the least considered. Indeed the domain level has been up to date the realm of interest of the artificial intelligence community. They have spent most of their time to transform the "continent of knowledge" into "an island" and yet were unable to do this.[9]

Hence I view Coplien's thesis as a blessing because he manages to widen horizons within the very context of a widespread object oriented language such as C++. He asks *How would we know to apply the object paradigm instead of another?Or, asked another way, how would we know, when doing analysis, to focus on object-oriented abstractions, instead of naively exploring the universe of paradigms supported by C++?* The object oriented paradigm produces a good architecture if the "pressure points"of the applica;tion line up with the dimensions of commonality and variability supported by the object paradigm (hot

[9] In my last Scientific Report [Lour00] to FAPESP I showed extensively that this activity requires the development of highly developed cognitive processes such as those that led Rilke to introduce the poetry-thing. His metaphysical reasoning under Rodin's influence was tamed and he realized the essence of art. He suddenly was beyond the dualism subject-object when he managed to write the famous poem-thing: The panther! Curiously Heidegger praised Rilke as a modern prophet but could not appreciate his poetry-thing. Being employed by Hitler, he could also not understand the similar transformations that modern art was undergoing with the introduction of the object or the thing in art. Art itself became a thing. Moreover the activity of software development is thoroughly linked to these sort of cognitive processes.

spots). The object paradigm presumes commonality in type and structure; it expresses variability in implementation. In a pure object-oriented world, other paradigms take second-class status and are difficult to express except in object-oriented terms.

Few designs are purely object-oriented because they express other important dimensions of commonality and variability.

C++ captures some of these non-object-oriented design structures well. Most C++ programmers consciously use some degree of procedural decomposition. FSMs (finite-state machines) are part of many object oriented designs and can conveniently be implemented in several ways in C++. Templates, overloading and other language features aren't intrinsically object-oriented. Moreover database constructs, parallel programming constructs, concurrency programming constructs, distributed systems constructs and other important solution domain structures can be accommodated only by convention in C++, if at all!

Basically Coplien conceives **multi-paradigm design** to go beyond objects to express design structures that find rich expression in C++. Hence other paradigms help us understand other important relationships that make a system a system. He shows C++ can express some of those relationships directly.

He also shows that different styles of managing commonality and variation correspond to what commonly are called paradigms in contemporary computer science.

Multi-paradigm design shows that the structure of a domain may be more complex than naturally can be expressed with a single paradigm. Furthermore it shows how to identify the latent structure of multiple paradigms within a domain. Such identification of paradigms offers design legitimization for the widespread programming practices that combine paradigms, but which are usually viewed as impure or bad design. Multi-

paradigm design offers a formalism based on commonality and variation that allows designers to express the intent of such implementation constructs in terms of design-level considerations [Copl00:263].

Kuhn [Kuhn70] popularized the term paradigm as a world model. This concept does not tune with the definition of paradigm adopted by Budd [Budd95], which is adaptable to the object paradigm, functional programming, rule-based programming and many others.

There is no general rule, one might for example say that object oriented programming in Smalltalk which is culturally normative may be a paradigm even in the Kuhnian sense.

I.3 Specification versus evolution

The founders of software engineering such as Djikstra state: *Our task is only to build the (formal) Machine to satisfy a given (formal) specification at interface a; this specification is a 'logical firewall'*. So according to Djikstra the software engineering's subject matter is located solely in the Machine [Dijk89]. Moreover Barry Boehm [Boeh00] adds: *The first Ability to Perform in the first Key Process Area in the software CMM (Version 1.1): Analysis and allocation of the system requirements is not the responsibility of the software engineering group but is a prerequisite for their work"* [PWCC95].

However W. L. Scherlis, responding to E.W. Dijkstra [Djik89] tries to extend software development : *Our task includes formalizing the Customer's Requirement and the Problem domain, and deriving a formal specification of behaviour at interface of the Machine.*

Michael Jackson [Jack01] asks *What, exactly, is in the problem domain? What, exactly, is the requirement?* He advances the principle of explicit description. It's too unsystematic

and error prone to make intelligible an unintelligible specification by suitable annotations to the specification or program. Domains (parts of the world) significant in the problem:

1) must not be partial shadows in the program's penumbra
2) must appear explicitly in the problem diagram
3) must have dedicated descriptions of their properties. The relevant domain properties (assumptions, rely-conditions) must be clearly stated and examined (e.g., behaviour of vehicle drivers, vehicle wheels and sensor interactions, how light units work, etc). Of course, one should avoid the infinite regression:

- Successive "Why?"questions have no stopping place.
- Why enforce one-way working?
- Why ensure orderly safe traffic?
- Why use cars?
- Why preserve human life?
- How to avoid an unbounded regression of goals?
- Any development has a specific customer
- The customer is some collection of directly interested parties
- The customer has some responsibility and some authority
- Customer responsibility and authority bound the problem…
- …as represented in the problem diagram.[10]

He concludes the model is not the real world. A model domain is a part of the undecomposed machine, designed by the software developer. It is a local variable of the undecomposed machine for a significant information problem. It is a part of the problem domain for two decomposed subproblems (building and using the model). Model domains are surrogates for their subject domains. The model and subject domain phenomena are distinct. Model domains are necessarily imperfect surrogates. The imperfections limit the satisfiable requirements. Therefore it is impossible, for such problems, to state a satisfiable

[10] Obviously in my knowledge system, the concern with the transcendent reasons is not excluded. My master's dissertation was clearly outlined concerned with them. It is entitled *Spirit, energy and information: essential elements of the urban ecosystem[*Lour88]. When considered in this broad conception, obviously paradigm here has a Kuhnian sense.

requirement independently of the problem decomposition and of the development of the model domain.

In other words, it is not possible to ignore the human dimension.

Moreover Jackson in his keynote at the last joint ESEC/FSE9 held in Vienna emphasizes too often software engineering is sequential, not "co-operative": *I observed the social consequences of this approach in several aerospace system-architecture-definition meetings…While the hardware and systems engineers sat around the table discussing their previous system architectures, the software engineers sat on the side, waiting for someone to give them a precise specification they could turn into code.*[Boeh00]. Yet Software Engineering must be a cooperative activity.

And concludes his talk: *The real problem of engineering education is the implicit acceptance of the notion that high-status analytical courses are superior to those that encourage the student to develop an intuitive 'feel' for the incalculable complexity of engineering practice in the real world* [Ferg92].

However Lehman and Ramil [LR01] as shown below are concerned with the questions stated above and introduce the **E-type programs** simply as ones whose acceptability depends on the perception, judgement and degree of satisfaction of appropriate stakeholder(s).[11] Software used to solve a problem or address an application in a real world domain is in general of this type. Such software is a model of a solution to some application in the domain of interest. Now both the application and its operational domain are intrinsically unbounded. No matter how many observations or properties are identified and associated with either one can always add another. The software, on the other hand is finite

[11] Curiously agile software methodologies explicitly agree that people are first-order components in software [Cock01]

in their viewpoint. Agile software methodologies try to overcome this problem seeing the software as a cooperative game of communication, discovery and invention [Cock02]. However efforts developed by the community interested in reflection try to build open computational systems [Stey94] and indeed my knowledge system can only be implemented in this context. So why does computer science community seem to ignore these more advanced efforts and does not thrive on them? Apparently because too much software is already written and they want to continue using it. In my viewpoint this happens because one is not interested in realizing sustainable development officially started with the Brundtland Report assigned by UNO in 1988. I mean if all artifacts were conceived within the context of sustainable development, soon every computer expert would unfold encompassing domain models that call forth open computational systems. If one disobeys, the implementation becomes untamable.

I.4 S-type and E-type programs

In the *IVth International Workshop on Principles of Software Evolution IWPSE 2001* held in conjunction with ESEC and FSE-9, Lehman and Ramil [LR01] courageously put forward a classification scheme, identifying two types of programs, namely **S-type** and **E-type**.

S-type programs are programs for which, by definition, the only criterion for successful implementation is that the program satisfies a pre-stated formal specification, that is correct, in the mathematical sense, relative to that specification. The S-type program definition implies that the specification expresses all the properties that the program is requiresd to possess to be deemed satisfactory or acceptable. With the exception of systems

where decidability issues arise, demonstration of correctness, by means of a proof for example is not a matter of principle but of mathematical skill and available mathematical tools.

The designation S was chosen to indicate the definitive role that the *specification* plays in determining required product properties. Pfleger assumed an alternative interpretation that S stands for static, one of the properties of the S-type that distinguishes it form the essentially evolutionary type E as described below.

The E-type was originally defined as "…programs that mechanise a human or societal activity…"(Lehman 1980). This initial description was subsequently extended to include all programs that "…operate or address a problem or activity in the real world…"To remain satisfactory, E-type programs must be continually changed and updated. They must be evolved. Hence the designation E.

S-type software accepts that the problem to be solved is fully understood once the specification has been completed. Thereafter, it is primarily the knowledge, understanding and experience of the implementers that drives the implementation process. Learning during the course of that process is largely restricted to determination of methods of solution, or of the best method, in the context of constraints applying in the solution domain. The nature of feedback in S-type program implementation is restricted. It may well be present at a **very low level of development,** as of requirements, design, code or documentation, but plays a secondary role, and is unlikely to dominate the implementation process.

Validation is important from a business point of view since it determines the likely acceptability of the final product. Technically it shifts responsibility to the client since, contractually the process has been terminated by satisfactory completion of the verification.

When, because of a deficient specification, refinement is required the process is abandoned and a new one based on a new specification is initiated. In practice, that new process may well take advantage of the earlier one. Nevertheless conceptually the development from an initial concept and the derived specification, consists of a series of open loop processes rather than a continuous evolutionary process. S-type program development restricts the scope of evolution to a **very low level.**

In the case of E-type systems, on the other hand, the problem to be solved relates to the real world. An application (or change to an application) to be developed and the domain (or change of domain) within which it is to be solved are not in general clearly and uniquely defined. Thee will always be fuzzy aspects and feedback plays a crucial role here. In relation to evolution, any initial fuzziness in development involves at least two separate and distinct aspects.

1. The first relates to the intrinsic unbounded nature of any application and its operational domain. Initially the latter is neither precisely defined nor bounded in extent and in detail. Such uncertainty is removed by a bounding process that determines the domain over which the application is to be valid, used and supported, or to which it is to be adapted to provide a satisfactory solution in some defined time frame at acceptable cost.

   However once the system is in operation a need or desire to extend the area of validity to regions of the domain or to details previously excluded will inevitably arise since the latter will become the irritants and performance inhibitors. Equally, the system must be evolved by feeding back to the implementation organization information needed to modify or extend the domain so as to satisfy newly emerging needs, changing constraints or changed environmental circumstances.

2. The second aspect is concerned with the boundaries of the system to be implemented. As anyone with experience in systems analysis, specification and design knows the list of properties and function that could be included in a system is potentially unbounded. It is always in excess of what can be accommodated within the resources and time allocated for system implementation. Hence, from the point of view of potential coverage, the boundaries of the final system are arbitrary. But, unlike those of the domain, once developed and installed they become solid. Determined, at any one time, by the installed hardware and software.

   A user requiring a facility not included within this boundary will, in the first instance, use stand-alone software to provide the required facility. It may be possible to couple such software tightly to the system for greater convenience in co-operative execution. But, however the additional function is invoked and the results of execution passed to the main system, additional execution overhead, time delays, performance and reliability penalties and sources of error are incurred. The omissions become onerous, a source of performance inhibitors and user dissatisfaction. The inevitable result is a request for system extension.

I.4.1 Model-like reflection: the ability to describe the phenomenal diversity versus the inability to reflect this into programs

Properties such as these make implementation and use of the systems a learning experience. The system is intrinsically evolutionary. Its relationship to the real world may be described as **model-like reflection. The program is a bounded, discrete and static reflection of the unbounded, effectively continuous and dynamic application domain.**

Lehman enunciates his First Law of Software Evolution: A sequence of releases transforms the system away from one satisfying the original concept to one that successively supports changing circumstances, needs and opportunities in a changing world. If conditions to support such evolution do not exist, then the program will gradually lapse into uselessness as a widening gap develops between the real world as mirrored by the program and the real world as it now is.

He concludes that system evolution is unlikely to have been determined primarily by human management decision but rather the ultimate determinants appeared to be dynamic forces due to the feedback nature of the software process.

Conclusion

What is certain is that the software process as variously practiced today is far from perfect, expensive, the source of delay in many computer dependent projects with the allegedly completed software products displaying major defects and deficiencies.

Moreover as new software technologies, Object Orientation, Component Based Architecture, UML, Java as example emerge they call for and suggest new approaches to software implementation and evolution. In the case of component based paradigms, Ramil [Rami01] adds even if the components are of type S, if the total or host system is used in the real world the whole system becomes of type E, it must be evolved.. He and Lehman discuss this in [LR00]. Ramil also believes that although they have not investigates the evolution of OO-based systems and they will do this when appropriate data becomes available, some already argue that in the long-term OO approaches may be even worse than traditional paradigms. He thinks it would be interesting to compare patterns of OO-based versus non-OO evolution. The field is emergent and we must have patience.

The net result is that software implementation and evolution processes also evolve. The fact that the new technologies are different in principle to other earlier practice and that no comprehensive scientific base of framework exists for software technology increases the need for processes to evolve on the basis of experience, emerging insight, inventiveness and feedback. Computer scientists are reluctant to explicitly accept the artistic nature of the software development [Cock02], [Naur85], [Lour01] simply because although cognitive processes are inherent to each human being's self, it is a qualitative jump that mankind as a whole still has to face, if the necessary synergies to build Paradise on Mother Earth are to be triggered. This requires a holistic vision. The overwhelming work being done in computer science is of a fragmentary, compartmentalized nature.

I.4.2 Process evolution and the hermeneutic cycle.

While S-type programs do not evolve and when they no longer satisfy their intended purpose a new program must be developed to replace them, in an E-type program any instance is transient, ephemeral. Once executed it is gone forever. It will normally have been preplanned in outline with details filled in as progress is made. Unanticipated circumstances and unexpected conditions, specification changes, performance problems, budget changes, for example are the norm and lead to process adjustments, adaptations and changes on the fly. Such unplanned changes, though error prone and therefore, in principle, undesirable are often triggered by observation of the results or consequences of past activity or by perception of what lies ahead. The consequence may be a change to the planned upcoming process activity or a need to backtrack or iterate. In any event there is a complex mixture of feedback and feed forward based on individual and collective

interpretation, intellectual judgement and decision by humans that will determine how to proceed. Whenever people are involved some degree of freedom exists; otherwise their activity could be mechanized. That freedom relates to what is done, what is not done and how the former is done. Hence the process can sensibly only be pre-planned and defined to a limited extent and to some arbitrary level of detail. It can only be enforced at a comparatively coarse level of granularity. Enforcement of a process specified at a high level of detail in specific circumstances (e.g. life critical, such as medical or aerospace software), but this can, itself lead, for example to defect injection, inadequate treatment of unforeseen circumstances, high cost or serious time delays while autorisation to deviate is obtained.

If any model is to serve a useful purpose it must reflect the process as the latter evolves, and the inevitability of process evolution has already been discussed. The process will inevitably evolve, not only through pre-planning *in vitro*, but also dynamically *in vivo*.

Where impetus for change comes form a need to adapt a process to specific conditions or circumstances, model evolution is a consequence of process evolution. This is so even though initial evalutation may be obtained by implementing, exploring, and comparing alternative changes in the model by enactment or otherwise before incorporating the selected change in the process. Where this is not done changes made to the process, whether premeditated or on the fly (something that should rarely, if ever be done) must be reflected in a change to the model if the latter is to retain its validity and value.

If, on the other hand, the pressure for evolution comes from recognition of a need for improvement, the processs model can play a seminal role being used to design and evalutate the change before implementation. However exploited, the information that drives improvement is garnered from observation and previous experience. Model evolution is

also feedback driven. The flow will be from within the organization, form other software developers and from process experts and practitioners.

The time relationships between model changes and process changes and the nature of the feedback loops that convey the interactions. For the process the key word is immediacy whereas for software there is in general significant relative delay in feedback.

Hence Lehman and Ramil want to put forward a theory of software evolution. They have been primarily concerned with the properties of the phenomenon, the what and why of evolution and the how of evolution.

I.4.3 The principle of software uncertainty as a consequence of inability to grasp quality or the intrinsic nature of the thing

Although they reach interesting insights while examining the software evolution as a phenomenon, their concern with the "how"of the evolution is poor. For them, the software concerned with E-type software evolution includes "*all programs that, when executed in a specified real world domain (the execution domain), solve a problem (or a set of problems) defined in and part of that domain.* [LR01b]

Both domain and programs are models of (an explicit or implicit) specification that is itself an abstraction of the real world domain of interest that includes the problem to be solved.

By definition, the requirement specification reflects all those properties of the execution domain that are required for acceptable solutions of the problem. That domain and the program will have additional properties not addressed by the specification.

By omission or commission, such properties are declared to be of no concern in relation to an acceptable solution. An incompatibility between such additional properties, one from each domain is, therefore, of no concern, as long as the real world does not change. Change

is however inevitable sooner or later. A property of either the domain or program previously accepted as of no concern may then block achievement of an acceptable solution. Worse still, what was previously regarded as an acceptable solution may no longer be acceptable. Thus, as the real world changes, one or more domain properties may become incompatible with the specification rendering the abstraction invalid. This inference leads into another *if as a result of changes in the real world execution domain, a specification is no longer a valid abstraction of that domain, the E-type program that models the specification may be unacceptable. They conclude* that "*the behaviour of E-type programs when executed is inherently uncertain, cannot be guaranteed to be acceptable.* This is a restatement of the principle of software uncertainty. Needless to say again, this is so because no effort to adhere to open computational systems is being made by the overwhelming majority of software developers. [12]

Below Katayama courageously recognizes the importance of domain analysis and an isomorphic mapping between all levels of software systems. However he does not really exploit the phenomenological demand. In theory, it sounds intriguing.

I.5 Evolutionary domains

Takuya Katayama [Kata01] explicits that in evolving software we need to propagate changes in a software development phase to its succeeding phases. That is, requirement changes have to be propagated to the specification and specification changes to design

[12] Gregor Kiczales seemed to be an exponent in the direction of open computational systems, exploiting reflective capabilities. He turned to Aspect oriented programming (AOP). Every computer scientist in his own way is realizing the importance of the domain modeling. Hence this turn to AOP is a shy recognition of its importance.

documents, and the design changes need to be reflected on its program codes. So evolution process, in general consists of multiple change propagation activities between two consecutive phases.

In this reasoning, the evolutionary domain is not just a set of final specifications, but it is a collection of final or intermediate specifications or their fragments which could appear in the evolution problem under consideration, together with an order relation and two operators on them.

He naïvely proposes an evolution relation which naturally will correspond to a variety of choices in general and it is the key to scientific and sound treatment of evolution practice to identify the relevant relation. The most important among them will be 1) functional augmentation that includes inheritance in object-oriented programming and methodology and 2) refinement corresponding to instantiation in parameterized module and application frameworks.

Obviously his theory may perhaps work for simple systems concerned with data. It is a nice work because it shows clearly the correspondence that all different levels must have to be concerned with and ease evolution. However when dealing with complex systems such as those that simulate the real world and are the concern of the object oriented community in general, he will realize that not all the elements of the knowledge models could be directly mapped into object-oriented concepts. The generalization-specialization hierarchies as defined in the knowledge domains cannot always be mapped directly to the object-oriented inheritance hierarchies. Mehmet Aksit et al [AMT00] warns that in general it is possible to implement an object-oriented application that provides correspondence to a domain knowledge hierarchy. However this may require the creation of additional structures and interactions because a one-to-one mapping is impossible.

I.6 How does the need for evolution affects the object-oriented community?

I have already started this discussion in my PhD thesis in the seventh chapter entitled Hermeneutic Computer Science: the hermeneutic nature of the Model of Primary, Secondary and Tertiary Waves to design and plan sustainable cities and a prototype based obj ct oriented programming language Self [Lour98].

Generally speaking the mechanism of inheritance based on delegation favours evolution . Yet the object-oriented community insists on sticking to class-based language.

Parallel approaches have been concerned with the need for evolution especially at the code and design level.

Curiously all of them recognize the superiority of dealing with delegation to ease evolution.

Günther Kniesel [Knie00] on the third chapter of his PhD thesis about behaviour evolution is concerned with the need for a model of object-oriented languages that supports both behaviour evolution and unanticipated extension as a highly desirable tool for every analyst, designer and programmer.

The class-based model on which most widely-used object-oriented programming languages rely cannot easily express changes in the structure or behaviour of an object.

In contrast, prototype- and delegation-based languages like Self can directly express changes of structure and behaviour but do not provide safety guarantees like static typing and guaranteed uniform structure of groups of objects, which are widely considered indispensable for production programming.

I would rather emphasize the difficulty of the computer scientists to mimic cognitive processes due to the strong emphasis on computer science as science rather than art.

Indeed, the computer scientists have been trying hard to catch up with the high complexity of the systems around us. The traditional attempt to grasp the thingness of the world through the class concept is still embedded in the Aristotelian logic [RC00]. Since it is not so expressive to describe the dynamic and evolutive nature aspects and to jump from analogical reasoning to hermeneutic reasoning is not smooth, the solution was to liken the class in an object-oriented program to a symmetry. Geometry is always the realm where more expressive ideas can be revealed. Indeed when a closed system encounters stress it loses symmetry. Many programming language constructs express symmetry. Since those constructs have been failing to solve design problems, programmers resorted to patterns to express this symmetry breaking in the programming language[CZ00]. A pattern is indeed a higher level of abstraction to grasp the thingness of things. The insistence in taming them to class concepts makes it difficult to exploit their true potential. They are better expressed in prototype based languages, that grasps better the hermeneutic reasoning. More and more programmers realize their delegation inheritance mechanism models reality and unanticipated changes elegantly and straightforwardly [Knie00], [Lour97].

Obviously the lack of more precise theoretic underpinnings may lead to the erosion of the object oriented paradigm that seems to grasp more faithfully the outlined trends above.

So it is current in the literature to put forward the mismatch between the objectivity of object-oriented languages (class and object-based languages) and the subjectivity of our problem domains. And an explosion of new ad hoc mechanisms is happening every time a problem that includes multiple perspectives has to be faced. And instead of perceiving that there is essentially nothing wrong with the thing or the object, because it is indeed a

successive multitude of inner and outer parts, all sorts of techniques are popping up to extend or modify object-oriented systems to support dialectic worldviews or "subjectivity [Knie00], [SU96], [TO93], [Bruj98], [HH90], [OT99], [ATB96], [Holl92].

The right attitude would be to describe all the levels of computational reasoning from domain analysis to implementation code endeavouring to penetrate the essence of the thing to be modeled in multi-paradigm design as proposed by Jim Coplien [Copl00].

The **agile methodologies** highlight the active participation of the stakeholders opening the gate to accomplish Peirce's idea of unlimited semiosis.

To realize that in a dialogue, in a game, in an experience, the human's way of being does not allow manipulation, the subjectivity of the players does not matter when they are really engaged in the process of playing. Agile software developers seek this goal. Now the reality of these events becomes significant. They have their own nature, independent of the awareness of those that accomplish them. Where no subjectivity limits the thematic horizon and where nobody behaves playfully, one meets the essence of these phenomena. However the true attitude of those involved is detached and joyful because they feel they are growing vertically to infinity and are able to extend horizontally infinitely. Tension is incompatible with the spirit of these phenomena. Patience and the natural flow of things cause them to bloom [Gada75].

Rene Thom, the founder of catastrophe theory, Gadamer, Rilke, Rodin, Heidegger, Peirce among many others strove to show that the rich nature of the thing not only encompasses all these trends but also transcends them. It seems the current trends in computer science are realizing this.

Briefly I introduce a description of behaviour evolution in prototype and class-based languages.[13]

I.6.1 Prototype-based behaviour evolution.

Object structure modification is the primary choice if fully unanticipated, identity-preserving changes of individual properties of individual objects are required. However, this ultimate flexibility has a high price. Property-level granularity makes keeping track of properties that belong together and are to be modified simultaneously, a tedious and error-prone task. As a further consequence, object structure modification is incompatible with modularity and conceptual modeling. When the code that manipulates the structure of an object can be scattered through the whole program there is no modularity anymore.

All aspects of object-specific functionality and dynamic change of behaviour can also be modeled by delegation.

Fully unanticipated changes are possible if object-identity does not need to be preserved, whereas identity-preserving changes have to be partially anticipated by providing suitable methods in the object that is to be changed. We call this partial anticipation, compared to calss-based systems, where changes have to be anticipated in the class to be changed and the class that represents its modified behaviour.

Compared to object structure modification, delegation is more convenient when multiple properties have to be changed at once. It allows programmers to choose between objects

[13] This does not mean at all I am in agree with the viewpoints exposed. But to be able to express my methodology that mimics a semiotic, hermeneutic and autopoietic "machine" (hence simulate the most evolved human cognitive processes) I have selected approaches that are trying to approximate this goal. Part II reflects the essence of my concerns. Although I am amazed at the "bottom-up" efforts, the current chaos or stagnation in computer science is curbing the experts to start thinking by themselves. To abandon paradigms related to general theories that have no meaning when faced with the stimulating changeable evolutive nature of the real world problems. And great progress is indeed being achieved.

that contain a consistent set of relevant properties, instead of forcing them to keep themselves track of all the properties that together achieve the intended functionality. The methods in the delegation-based variants are significantly simpler. Because most conceptual changes of an entities' behaviour involve consistent changes to a set of interrelated object properties, delegation directly supports conceptual modeling and modularity.

Unlike structure modification, delegation enables sharing of properties among multiple objects, which in turn enables modeling of objects that can be regarded from different alternative point of view, or different versions of an object.

The major conceptual restriction of delegation that prevented its use outside prototype-based languages is its claimed incompatibility with static typing. Like object structure modification, delegation is typable only under major restrictions. Especially no system that includes subtyping and delegation is known so far.

Günther Kniesel concludes:

1. Object structure modification as well as delegation effectively support unanticipated behaviour evolution.
2. From the point of view of typing there are no clear arguments in favour of either one of both alternatives.
3. From the point of view of conceptual modeling, modularity and maintainability delegation is by far superior.

Therefore in the end delegation is the preferred behaviour evolution mechanism to be integrated into a unified model of class- and prototype based systems. The apparent incompatibility with static typing and subtyping is a problem that is to be solved as one step towards the unified model.

I.6.2 Behaviour evolution in class-based languages and design patterns of the GOF[GHJV]

Traditional class-based systems do not effectively support behaviour evolution.

The class-based model cannot easily express changes in the structure or behaviour of an object. Required behaviour evolution has to be hard-coded into application programs.

Because they have no object structure modification operations (which would contradict their essential property, creation of objects with fixed structure by instantiation), class based languages essentially model object specific properties and their dynamic change by trying to mimic delegation.

There are two basic scenarios in class-based variants for modeling anticipated and unanticipated change by simulating delegation. These scenarios can be regarded as meta-patterns that distill the technical essence common to various design patterns.

The earliest movement spearheaded by Jim Coplien introduced the design patterns fruit of the inspiration on the *A Pattern Language* from Christopher Alexander as a way of introducing change and adaptation in class-based languages[GHJV95].

According to Kniesel, the flyweight, state and strategy pattern and bridge, decorator, chain of responsibility and proxy pattern can be distilled as meta-patterns to simulate delegation.

He shows how to do this. However these meta-patterns do not achieve full functionality of delegation.

Because the functionality missing from the language is simulated by a set of cooperating classes, these classes tend to be tightly dependent on each other in ways that are not grounded in the application but just in the technical details of the pattern.

Such additional dependencies and assumptions built into the design require consistent changes in a complete hierarchy of classes even for small unanticipated changes in the application logic, e.g., addition of a method to a class or addition of a new class. Thus the application of design patterns for simulating behaviour evolution can impede reuse (of existing code and designs) and complicate program maintenance, achieving the opposite of what is widely regarded as a main benefit of object oriented programmingl

Therefore extension of traditional class-based object models by a mechanism for unanticipated behaviour evolution is ultimately required.

He concludes that delegation is the preferred behaviour evolution mechanism to be integrated into a unified model. The reconciliation of delegation with static typing and subtyping is a problem that is to be solved as part of the integration process. He extends **Java** with delegation and the outcome is **Lava,** that can express any conceptual object evolution required by a statically typed application.

Likewise Jecel Mattos de Assumpção, one of the most active members of the Self community tried to make clear that design patterns implemented in a prototype based language like Self become unnecessary due to the fact they become idioms in Self. Below I reproduce briefly his viewpoint.[Assu00][14]

As promised, here are some quick comments from a Self programmer's viewpoint on the patterns in the "Design Patterns" book by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. The numbers in parenthesis are the page numbers where each pattern is described.

I hope this helps,
-- Jecel

Creational Patterns

[14] I attended a tutorial given by Frank Buschman at ECOOP'01 entittled Patterns at Work. He agrees it is possible to implement design patterns and architectural patterns in Self but he strongly argues that one should favour C++ and Smalltalk because lots of efforts were done to make these languages efficient and safe!

===> Abstract Factory(87):

I have suggested that it might be interesting to add abstract types to Self in the form of interface (to use Java terms) objects. This would be a good use of this pattern. Instead of writing

a: list copyRemoveAll

which refers to a specific prototype (list) directly, we could have

a: interfaces sequence getOneFor: 'fifo'

Of course, with dynamic types this pattern is considerably simpler in Self than in the book. I can't think of an example where this is used in Self 4.1.2.

===> Builder(97):

This goes against the idea of creating new objects by simply cloning an existing one. Of course, sometimes cloning is anything but simple. See the copy method for morphs, for example. But it is probably better to stick with the current Self style of stuffing all the complexity in the 'copy' method instead of creating separate "builder" objects.

===> Factory Method(107):

It is hard to see what the fuss is all about when you have dynamic types and cloning. Just put a prototype of the "concreteProduct" in a slot in the prototype of the "concreteCreator" and patch the copy method to go down one level, here. See the low level graphical framework in Self (canvas, windows, graphics contexts...) for an example. No big deal.

===> Prototype(117):

*This *is* a prototype based language. You can't avoid using this pattern if you want to! No big deal.*

===> Singleton(127):

All objects in Self are singleton unless they have traits clonable (or some suitable replacement) as one of their ancestors. No big deal.

Structural Patterns

===> Adaptor(139):

*This is *so* much simpler with implicit, dynamic delegation (data parents). When wrapping an objects with many methods, you only have to worry about the ones you will need to change, not about all of them. And Self really makes this pattern more powerful by allowing you to override data slots as well as method slots (even better: you can override a data slot with a method slot and vice versa). The only thing to watch out is how to handle copying and how to avoid having the adapted object's identity "leak out". Reflection might help with these problems. I don't know of any examples in Self 4.1.2.*

===> Bridge(151):

You could use implicit delegation for this pattern, but the example in Self (canvas) uses explicit delegation instead.

===> Composite(163):

Morphs are a great example of this. I am not sure that any Self features help very much, here, other than having dynamic typing make what objects can be plugged into what others more flexible.

===> Decorator(175):

It would be more pleasing, in my opinion, to use data parents (implicit delegation) for this pattern. I don't know of any examples where it is used in Self 4.1.2.

===> Facade(185):

I don't think Self makes this pattern simpler than in other languages. I was going to say that 'desktop' could be an example of this, but that is pushing it a little....

===> Flyweight(195):

I would say that Self's advanced inlining compiler technology makes this pattern less necessary than in other systems. This pattern is also used as a classic example of the advantages of reflective systems (see the Open Implementation pages).

http://www.parc.xerox.com/spl/projects/oi/workshop-94/foil/main.html

Oddly enough, there are examples of this in Morphic. There are several cases where the strategy pattern should have been used but the

information was represented as simple integers instead (see alignLeft, alignCenter and alignRight in columnMorph and similar options in related morphs). This is a design bug - I think the flyweight pattern should never be used in Self.

===> Proxy(207):

Exactly the same comment as for Adaptor. Except that there are plenty example of this in objects named, oddly enough, "proxies".

Behavioral Patterns

===> Chain of Responsibility(223):

The event handling in the user interface partly uses this pattern. Having implicit delegation might be nice, but I am not sure it would make much of a difference, here.

===> Command(233):

Event objects in the user interface could be considered an example of this. I think that this is nicer to do in Self since making lots of objects that are a little different isn't as bad as when you have to create a different class for each one.

===> Interpreter(243):

The only example that comes to mind is tinySelf 1. While the Self language itself doesn't make this pattern much simpler, being able to develop and interpreter in a dynamic user environment really helps. I think that most cases where this pattern might be used could be better implemented as a parser generated by Mango instead.

===> Iterator(257):

As Smalltalk Stream classes show, this pattern is much easier to implement in dynamically typed languages. I don't know of any examples in Self 4.1.2 and sometimes miss them a lot. My students implemented iterators for matrix objects (for an unfinished 3D GUI project) and that made a lot of the higher level code considerably simpler. Iterators are important in languages like Self and Smalltalk that have a very awkward syntax for accessing vector elements.

===> Mediator(273):

Hmmm... I am not sure, but the damage/redraw logic distributed between

morphs and the world Morph might be considered an example of this. I don't see that Self is particularly helpful in this case.

===> Memento(283):

Don't you just pity the poor slobs who have to program with systems that can't save Snapshots? :-)

You don't need this pattern with Snapshots, but in Self we do have it in the form of the Transporter. It would be nice to replace these two great tools with a real persistent object store.

===> Observer(293):

I would say that the damage/redraw code also includes an example of this pattern. Self doesn't help with this either, but a reflective layer might.

===> State(305):

Who needs this when you can have data parents? See the various tree objects in Self 4.1.2 as an example of why we can live without this (or as an example of how to implement this with implicit delegation, depending on your viewpoint).

===> Strategy(315):

This can also be implemented as data parents, but I have used explicit delegation in my own programs instead. In the latter case Self doesn't bring much to the party.

===> Template Method(325):

This is used in many places in Self, including in the C++ implementation of the virtual machine. For some reason I tend to think of the Beta language when I look at this pattern, but it is easy in Self as well.

===> Visitor(331):

I can't think of any examples of this in Self, though some of the runtime code generated by Mango might qualify.

Likewise other representative groups from object-oriented community are criticizing strongly the solution offered by design patterns.

As I have already pointed out in my PhD thesis, design patterns are inspired by *A Pattern Language* from Christopher Alexander. In section II, I hope it becomes clear to the reader what is faulty with this approach in terms of modeling. His greatest merit is the link to the phenomenon. However this link cannot be one of description only. Moreover Alexander does not geometrize his conceptions I mean he does not create the underlying geometric model to support his conception. He is trying to pursue this in The Nature of Order. Jim Coplien and Richard Gabriel strongly favour this Alexander's search. Richard Gabriel's book *Patterns and Software: Tales from the Software Community* contains what is probably one of the best descriptions of Alexander's search for truth and beauty in carpets and use of the bead game. Alexander has moved out of culturally sensitive patterns into cognitive and psychological aspects of geometry which are universal. He is concerned with unlocking the secret behind what constitutes the essence of what he now refers to as "wholeness'. What is it that makes some carpets more "whole" than others? *The Nature of Order* is a kind of grand unified theory on how things grow in nature (natural processes). What is regrettable here is that he is not linking the design patterns to this geometric concerns with the aim of shaping a strong modeling. However Coplien views things with different eyes and stresses enthusiastically:

So the software patterns movement is still in its infancy (we still have problems writing and defining patterns and pattern languages) and we may very well have to sweat through 20-30 years of software patterns and pattern languages before the industry is ready to progress and evolve as a community to where Alexander has taken patterns now with The Nature of Order. [Copl97]

Recently Coplien and Zhao [CZ00] argued that the design pattern language from the GOF forms a contextual framework for the formalization of symmetry breaking (the very

definition of individual patterns) and that such a framework forms not only a language, but also an algebra of specification. The outcome seems likely; after all, a geometry is essentially an algebra of symmetries. The stated purpose of a pattern language: to create whole systems through a piecemeal growth process. This is essentially in tune with what will be shown explicitly in the chapter II. Phenomenal diversity modeling. Yet the problem with design patterns remain once they mimic reality directly. Phenomena do not speak by themselves. They must be subsumed under regional or local categories and these categories must reproduce the internal diversity of the phenomena how I show in Part II.

Maybe the strong criticism it is receiving by the OO community is due to this.

I.6.2.1 More criticism on pattern languages

This criticism is intense and address many levels, which is beyond the scope of this Scientific Report to detect.

Design patterns represent stylized ways of solving commonly encountered problems in the design and implementation of software. Besides the famous GOF design patterns, there are the architectural patterns [BMRS00^{+}]

And all the others that are being created through the famous conferences on Pattern Languages of Programming.

Despite the similarities between patterns and well-defined language features, like Jecel Assumpção Jr shows above, and this happens with many other programming languages as well such as CLOS and C++ and Smalltalk, the patterns are intended to describe solutions in the absence of "unusual language features".

One of the first strong criticism on patterns comes from Peri Tarr and Harold Ossher, the authors of subject-oriented programming [TO96] . The goal of Subject-oriented programming is to provide unusual extra-languages features for solving a number of problems that arise in software composition. These features are provided so taht solutions for some commonly encounters problems can be expressed using the already usual idioms common in object-oriented languages. Like those generously provided by delegation. The subject-composition approach makes it possible to remedy several of the problems cause by the patterns, including: indirection/confusion, preplanning, object schizophrenia and hierarchy hardening. They discuss the problems and analyse them in each design pattern.

Composition filters from Mehmet Aksit et al [AB01] was created because they view patterns as not able to address composition and separation of concerns.

Next chapter I analyse how a modeling based on phenomenal diversity should be. Obviously the reader shall conclude this approach belongs to E-type.

On chapter II I analyze what it is modeling based on the phenomenal diversity.

On Chapter III, I introduce aspect oriented programming [KLMM97^{+}] that is now sheltering all the approaches concerned with composition and separation of concerns.

Likewise to solve the current problems introduced by the failure of the programming languages to cope with the expressiveness and evolutive nature of the real world, I would suggest a similar approach to validate semiotic, hermeneutic and autopoietic methodologies or guide the others towards this trend.

The **agile methodologies** may serve this purpose very well. While these questions applied to aspect oriented programming may lead to interesting conclusions.

I hope this introduction will enable the reader to follow without difficulty the sudden changes happening in the realm of computer science. Instead of being scared, I hope he/she will embrace the trend, triggering the creative cognitive processes inside each human being. This is the necessary measure to promote synergies and improve dramatically the quality of life on Mother Earth, building a sustainable planet.

II. Phenomenological diversity modelling

In the long run more and more breakthroughs in the object-oriented paradigm ease the task of building an isomorphic software model to my ecodesign model.

This brief introduction aims at giving you a glimpse informally due to time concerns of how the so-called **heavy methodologies** that are a far cry from my ideas are not only trying to become more expressive but also seeing the emergence and fast proliferation of the so-called **agile methodologies** . The latter obviously tend towards my beliefs. Indeed generally speaking there is an effort to mimic or simulate real world. as Manny Lehmann put forward in a recent e-mail dated October17, 2001:

Re OO, not quite sure what Juan said or meant but we will b meeting Friday week so that discussion will be on the agenda. Meanwhile let me just add that I have for many years regarded our studies of software evolution as a special case of more general systems evolution, as the fruit fly - drosophila melanogaster - of general systems evolution., the fruit fly because its being non-physical, its rate of evolution is so much higher than that of a biological/botannical system, a vehicle, a weapons system or a city for example. Thus I would expect many of the high level observations to apply although the detailed patterns and rate of evolution will be quite different. Thus the E-type concept applies to all such systems in the sense that, obviously, they evolve and their evolution is strongly driven and constrained by feedback effects and by real world interactions. In this wider sense, too, Systems based on OO concepts, approaches and technology are also of type E (even if their components are type S, ie formally and "completely" specified). I think that what Juan meant was that we had not studied, did not in fact have, any data on OO systems. But will know more after our next week discussion.

Paradoxically independent of the concern with simulating the real world, the so called generalized-procedure (GP) languages including procedural languages, class-based languages and functional languages (common root in that their key abstraction and composition mechanisms are all rooted in some form of generalized procedure) cannot be

recognized as true semiotic machines in the sense put forward by Winfried Nöth [Noth02] and addressed in the introduction. A closer exam however is necessary both to check if they belong to E-types or if they are semiotic machines.

However there is still a gap between the urgent needs mankind faces towards building autopoietic systems/machines that would have a great impact on the quality of life on earth and the shy attempts towards viewing software as a cooperative, inventive, communicative game. Figure 1 sketches coarsely the state of the art in computer science. The farther to the left more one is in the realm of formal computer science, the farther to the right more one is involved with hermeneutic computer science.

Briefly the latter is concerned with the way art is. More and more computer scientists have been likening software to an artistic activity.

Husserl, Heidegger, Gadamer, Vygotsky, Foucault, Wittgenstein (last phase) are concerned with the hermeneutic reasoning in philosophy. Their contemporary counterpart is Maturana, Varela, Prigogine, Gell-Mann. In computer science, Richard Coyne, the Dreyfus brothers, and the post-artificial intelligence research from Winograd and my research.

It implies the condition that all levels of the knowledge system will mirror the domain model. This begs the question: what is so special about my ecodomain model? As figure 1 puts forward, it is special because it is structured. I will try to make the reader sense how agile methodologies are tending towards structuring rather focusing on the concept of the structure category due to the need of finishing this Scientific Report according to he planned schedule a year ago!!!

René Thom and Jean Petitot-Cocorda [Coco85] argues that a model is an analogy between a phenomenon X and a built object M (the model) . Since the latter simulates X, it enables

us to answer a question Q' concerning the phenomenon X or to reproduce its intrinsic nature. For the model M to be legitimate, it is necessary:

1) that the question Q' determines the building of the model or that we manage to reproduce the intrinsic nature of the phenomenon
2) that we manage to translate the question Q' to a question Q concerning the model; this demands the need to control the analogy X-M between a phenomenon and an object (theoretical-formal) built in a certain language (justification a priori)
3) that the answer R given by the model to the question Q concerning M be submitted to an experimental verification, after having been translated to an answer R' to the initial question Q' or in case of the reproduction of the phenomenon's nature, to simulate it virtually or in reality (justification a posteriori by confirmation/refutation)
4) that the model having to be also explanatory deal with processes among invisible entities, yet enables the manifestation of the immanent aspect or visible morphologies.

The latter condition is concerned with the issue called phenomenological demand. In sustainaible cities the phenomenological demand is to understand how function transforms into form and then urban design into planning. Obviously these processes are intertwined. And the germ for planning is inherently coupled to the mechanisms underlying the substrates. Hence to understand how form and structure emerges from these mechanisms is a challenge. Here a circular reasoning or recursive reasoning is involved. But this link between the unity and the general must be expressed.

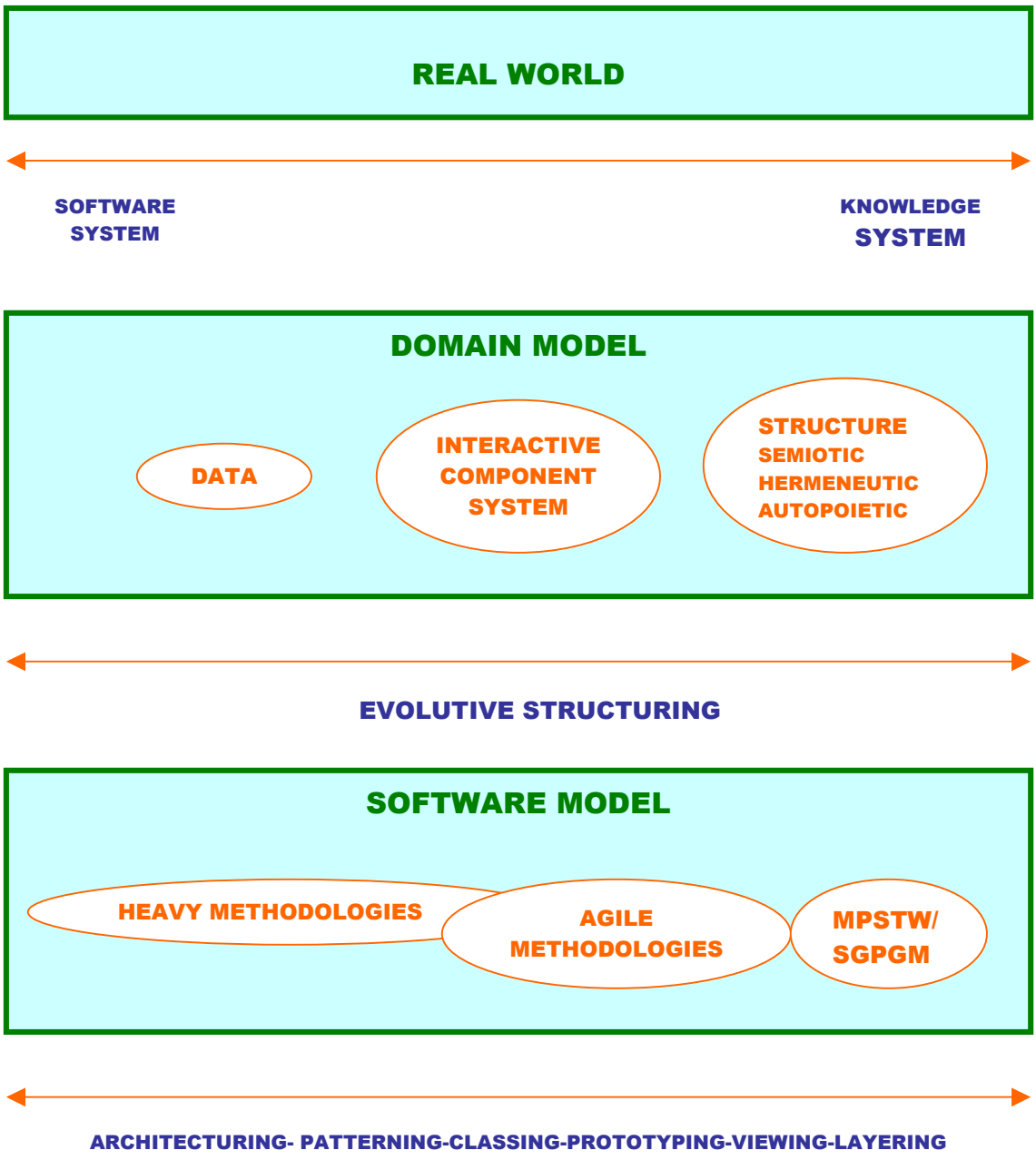


Figure 1. The farther to the right, the more the system approximates a semiotic machine.

In general the phenomenological demand is hardly ever perceived, although it is of the utmost importance in the understanding of the relationships between explanation and description.

Explanation refers to the underlying mechanisms, expected to be explanatory, derived from general laws and hence able to be translated into generative formalisms.

Description tries to mimic the visible nature of the phenomenon such as it manifests, without being concerned with invisible entitites or processes. To the contrary, to explain has been curbed to be understood as grasping the nature of the phenomenon, thriving on things that have no link to its phenomenological manifestation. This scission between theory and phenomenology is a fundamental characteristic of reductionism and has prevailed in mainstream formal computer science and exact sciences.

Hence if until now to grow, these sciences have sacrificed everything that is concerned with morphologies, structures and the phenomenal appearance, now time is ripe for them to understand how these emerge from the substrates. Indeed Peirce emphasizes that he also realizes that the sensory transparence in aesthetic consciousness of the general order of the universe found in the process of its development cannot be subsumed under the very being of the General. In his philosophy of continuity perception and its objects are to be conceived on the one hand as the limiting case of rationality, while rationality itself is to be conceived on the other hand as an object of sensory perception [Apel81].

A Peircean vision such as an object to be esthetically good, must have a multitude of parts so related to one another as to impart a positive simple immediate quality to the totality seems an unreachable ideal for programmers today. Hofstadter explores these reflective loops so well Bach's fugae, Escher's xylographs and Gödel's theorem . What was

liminary in Escher's tilings is today transdisciplinary and hence mathematically meaningful.

The essence of evolution seen by Peircean eyes resides in the category of infinite and continuity and real generality primarily related to the future in which it acts as a regulative principle of human actions to guarantee the completion of the universe's real lawfulness. Moreover Peirce highlights: *The very being of the General, of Reason, consists in its governing individual events. So, then, the essence of Reason is such that its being never can be completely perfected. It always must be in a state of incipiency, of growth. It is like the character of a man which consists in the ideas that he will conceive and in the efforts that he will make and which only develops as the occasions actually arise…Under this conception the ideal of conduct will be to execute our little function in the operation of the creation by giving a hand toward rendering the world more reasonable whenever, at the slang is, it is "up to us to do so".*

Hence the issue of the *a priori* justification of the models choosing a language which describes and controls the analogy M-X, their ontological bearing and fitness to reality of the phenomenon needs to be examined within the context of justification criteria for models such as: rational coherence, fitness to the experimental data, unicity, minimalism, power of change and evolution, etc. The former is the most difficult to be accomplished. Its lack is responsible for the fallacious evidence that introjects in most of the software systems: in the hierarchy of the organizational levels in reality, the levels are not autonomy and the base level determines causally the superior level. In the software system, the programming language tends to organize all the software. Only recently they are perceiving the importance of the domain model. And of course since the bottleneck is indeed the human cognitive processes, no surprise that now the human beings are first-order components in

software development [Cock01]. Obviously it is a sound reaction against the tyranny of the so called heavy top down methodologies prevailing in software development.

So like Peirce, Cockburn perceives the richness of human processes. And the challenge is to deal with reflective loops that link the individual to the general and the reciprocal. This is the key to evolutionary reasoning, embracing change and unpredictability.

This is hard to accomplish.. Especially in a world where each individual strives to survive. The world of artifacts around us apparently lead us to compartmentalization and fragmentation in reasoning. However the artists of the thing since Marcel Breuer have been striving to tell us they are also art. And indeed within the context of sustainable development, any green product defines the ecosystem and is defined by it. The idea is to reach zero emissions and hence mimic nature.

Of course it seems software developers are interested in promoting sustainability and boosting productivity. Hence they attack the heavy methodologies based on engineering disciplines that continue stuck to Brown Agenda and the fragmented way of seeing things. A panel *entitled Anarchy vs. Monarchy: A battle over the role of process ceremony and rigor* to be held at next OOPSLA'01 in Florida in October will try the appropriateness of the agile process movement. Determining which process model is the right one for your project is a critical factor in success.

.More and more the software community is realizing its role as producers of intellectual artifacts. At least more human practices introjected to them shape their gist. The ultimate challenge is to perceive that human nature and the nature of things around us are rather similar especially in terms of information. And again in an invited talk at OOPSLA'01 entitled *Harnessing Convection Currents of Information* Alistair Cockburn enables us to work with what is all around us, but we may not have good words for or do not know how

to alter. He advises us to reexamine software development as a cooperative game of invention and communication organizing the convection currents of information, adding information radiators and eliminating drafts, to improve the rate at which we develop.

Another panel: How do requirements relate to objects? At OOPSLA'01 the problem between artificial intelligence versus object oriented software will be tackled. While the artificial intelligence community starts to react and perceive a knowledge system as a software system, few see the reciprocal is true.

In the field of requirements engineering, object-oriented modeling is often considered as a means to create and represent a requirements model. Little attention is given to the problem of domain modeling in object oriented programming.

Requirements are just given textual input in the form of a short problem statement. The Workshop Report written by the participants of the Workshop on OOAE from ECOOP'01 [MG01] reflects this trend. However the fact that there were many heads thinking in the middle of the workshop almost a collective beheading occurred. The text reflects nicely how we managed to harness convection currents of information to air the issue of building evolutive software architectures without imposing no one's particular methodology be it agile or heavy!

Another invited talk: *Software transparency and object technologies* delivered by L. Peter Deutsch calls for transparency in software. In this context one realizes how wise the organizers of the Workshop I joined in June. Science calls for unbiased views. This leads to transparency. Deutsch defines software as transparent to the extent that it is conceived, developed, documented, licensed, distributed, and cared for to intentionally facilitate reading, understanding, analysis, validation, confidence; repair, adaptation, extension, evolution; interoperation, integration, incorporation; sharing and use.

Here Deutsch puts forward software transparency in all its aspects as the key to the usefulness of software over its lifetime (he also developed Smalltalk-80 programming systems). Above I stated that the key to evolutionary reasoning, embracing change and unpredicatability was to deal with reflective loops that link the individual to the general and the reciprocal.

Moreover at the beginning of my paper entitled *An evolutive architecture reasons a semiotic, hermeneutic and autopoietic entity* [Lour01b] I highlight the gist of transparency as: *To build an evolutive software system means to unravel something that happens or that we want to happen in the world as information insofar this is aligned as an "evolutive white box" available to a wide gamut of intelligent systems with manifold and differentiated cognitive abilities in the broadest contexts such as humans and machines.*

Obviously unsatisfied with the heavy methodologies or top-down methodologies that do not thrive on the observation of phenomenon as I wrote above and being unable to really tackle the nature of the thing, the software community on the one hand thrives now on the observation of phenomena or better on human experience.

Christopher Alexander writes: *We hope, of course, that many of the people who read, and use this language, will try to improve these patterns – will put their energy to work, in the task of finding more true, more profound invariants – and we hope that gradually these more true patterns, which are slowly discovered, as time goes on, will enter a common language, which all of us can share.* In A Pattern Language, xv.

Although the software community inspired on his patterns and developed the software design patterns,

Richard Gabriel [Gabr01] evidences the lack of a global vision in *A Pattern Language* . Almost all of his patterns have to do with how people live in homes, towns and cities with

other people and alone indulging in culture or savouring spirituality. But forget about the physicality of the architectonic object and its interaction with the environment. How this contributes to preserve the ecosystems. Either one emphasizes the ecosystem or the humans. However humanity, thingness, bioregionality must be intertwined to cater for natural evolution.

Brian Foote insists on the fact that patterns are about what works. They give us a way to talk about what works. He cites Stewart Brand in his book How Buildings Learn to *tell the story of a brilliant but lazy college planner who built a new campus with no sidewalks at all. She waited for the first winter and photographed where people made paths in the snow between the buildings. The next spring she put the pavement there. Patterns have this quality.* [MRB98].

So why did Alexander feel so attracted towards my ecodesign model and I feel his A Pattern Language has things that lacks in my model?

The real difference between the two models is the way they were conceived. While I first became interested in architecture I was seventeen years old and I would already not divorce it from a strong communion with nature. I had a deep background in Biology as well as in Literature and Psychology. These interests were already very strong at the age of twelve! Being disappointed with the lack of scientific reasoning in Architecture I made up my mind to follow Physics. But then I was warned about the trend of the physicists to create top-down monumental methodologies due to their lack of contact with the phenomena. They could only see the laws. Then I made up my mind to follow Biological Sciences. I was soon introduced to Ecological Modeling. Here the rational coherence was a must! If one must say something that applies to a domain of the real world, exactly what is done in modeling, rationality requires coherence about what we say about the objects or phenomena and their

translation into the chosen descriptive language (external coherence). These two aspects of rationality are in reality very difficult to satisfy with rigor.

For example, what's wrong with Alexander's and patterns's procedure seen from this viewpoint. First, one should study the thingness of the thing by itself and then to observe how this thingness interact with the ecology of the human behaviour. It is easier to make this explicit through a sink, a refrigerator and a stove. If we respect the ecology of the human movements, one would want the human being to walk the least possible among the three to accomplish the roles of washing dishes, preparing food and cooking. The industrial designers are the experts here and say we should lay out the furniture insofar as one walks in a triangle with the least perimeter. Well if you put furniture without caring about the heights of the human beings or ergonomic relationships waiting for when the house is built to see how human beings move around there, obviously this won't work. Le Corbusier reported that an average housewife would walk 13 km inside a house every day!! Moreover housework is compared to middle and heavy effort. So obviously design is a simultaneous and interactive game between things and human beings!

So how to simulate this game efficiently and playfully?

II.1 Structure

The main issue here is the choice or creation of a language that expresses the analogy phenomenon-model. Indeed this is the master pillar of all *a priori* justification. It cannot be derived simply by induction and abstraction based on the empirical data. Kant insists while undetermined objects of an empirical intuition, the phenomena do not speak. They speak

only when transformed in objects. This objectivation is a semiotic and conceptual construction derived from theoretical imagination.

There is data about the phenomenon (dependent on perception) and the reality or being of the object (independent of perception) as if the phenomenon were always introjected in their sense of object. It happens as if the basic theoretical language was empirically decided.

Hence :

1) a sense of object is determined by a system of regional categories. To subsume the phenomena of the considered region under these categories transform them into experience objects.
2) Paradoxically based on the principle that states the conformity to the things themselves, a model may not refer directly to the phenomena, but only indirectly (mediating it) through the categories that subsume them (put them under a general principle). This factoring likens to its rational legitimation. But we have been insisting on analogy. Hence the models must be models that reflect the diversity of phenomena. However the subsumption under the categories leads paradoxically from diversity to the unity of concept.
3) The models must deploy an internal diversity due to the meaning of the regional categories. The words that describe these categories must be tailored in such a way to fit different contexts equally well. Thom insists that here one must replace the meaning of the regional categories by an explicit and mathematical construction.
4) Indeed the geometrisation of concepts. One must spatialize the concepts insofar as to employ resources from geometric description – only this allow for true objectivation. However one can transcend this geometric space and uncover diverse

semantic spaces even more abstract where the concept can live!! This geometrisation enable the reduction of the transpatial caracter of the concepts and to control the analogy between model and reality. It is of the utmost importance for a model, not only its fitness to empirical reality, but also his ontological bearing (its conformity to an objective essence!). This step is known as schematicity. Kant defined it as construction of a concept in a mathematically determined intuition [Coco92].

Here one perceives the aim of the constitution of the morphological-structural objectivity. It changes the structural theories of theoretical types.

Hence *A pattern language* from Christopher Alexander "paste" too much to the empirical diversity without deploying categorical contents, and worse he does not manage to geometrize his concepts. However he is trying to pursue his research uncovering geometric patterns in his The Nature of Order [Copl97].

Only when we develop these contents both categorical and geometric, one can model the physical reality with precision and perfection. Thus the agreement between justification *a priori*, representing schematicity and justification *a posteriori* representing the experimental confirmation is the moment of legitimation of the models (modeling) (Figure 2).

In the Manifesto for Agile Software Development (http://www.AgileAlliance.org), seventeen software developers ranging from Kent Beck, Alistair Cockburn to Dave Thomas are uncovering better ways of developing software. They value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation

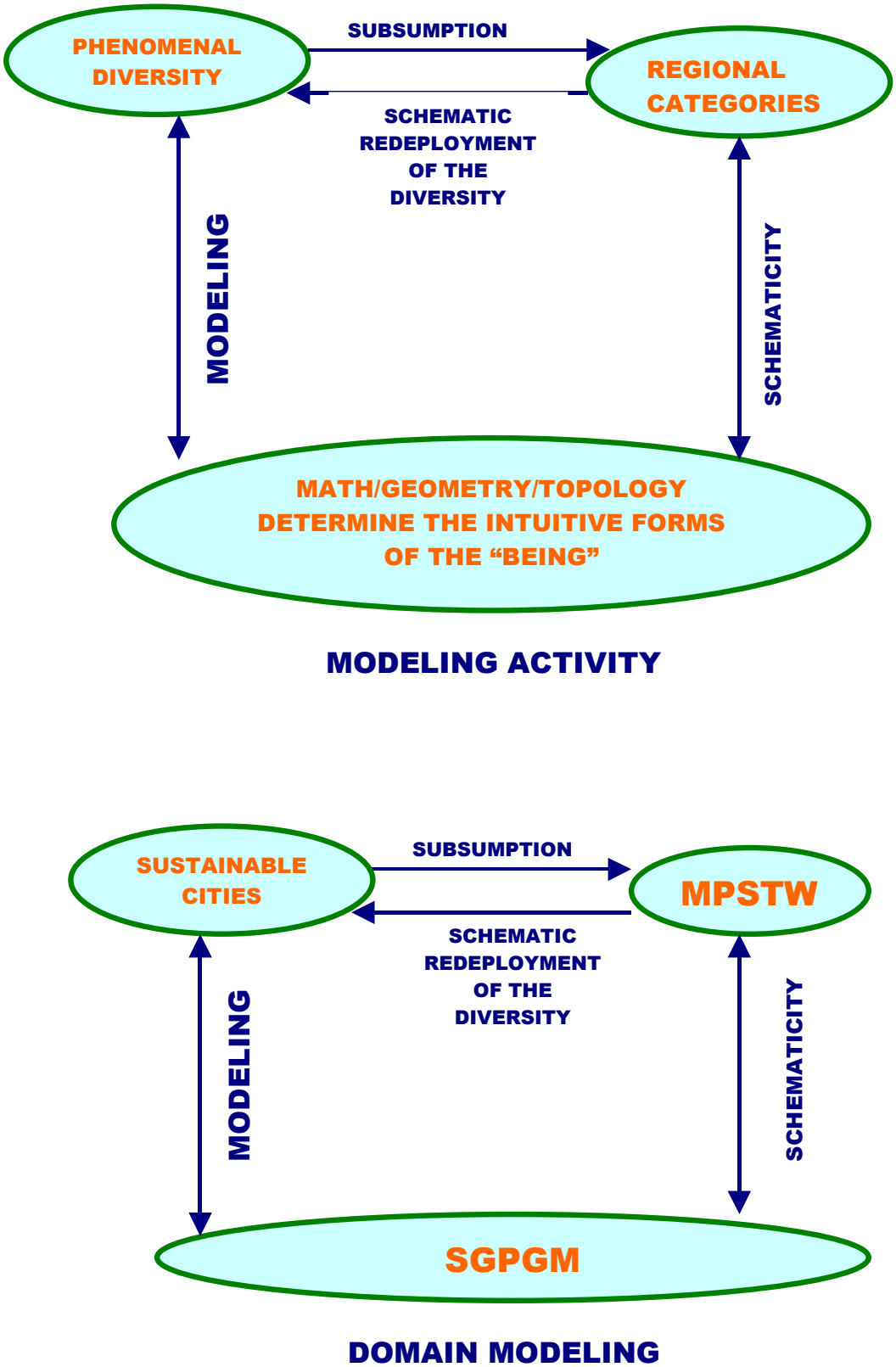


Figure 2. Building an analogous model to the phenomenal diversity.

- Customer collaboration over contract negotiation
- Responding to change over following a plan.

Obviously what they value is also highlighted more in my semiotic, hermeneutic and autopoietic approach.

What's the main difference between my knowledge system and the current software development (agile or heavy methodologies)?

The main difference is that its structure must unfold conform to the urban ecosystem. Ideal, non material structures with abstract forms of organization not reducible to systems of components in interaction pervade the urban ecosystem. Its essential elements are spirit, energy and information. All phenomenal diversity emerges from them. Since the domain model is analogous to the urban ecosystem that is being modeled, obviously the other two levels of a knowledge system, the architectural level and the code implementation level must be isomorphic to it and hence among themselves.

Obviously the agile and heavy methodologies are rather systems of components in interaction. However the approaches adopted by the light methodologies are likely to develop in the near future into highly structured approaches.

Several heads thinking trigger synergetic mechanisms. This overall synergy will necessarily lead to open trails in the dark jungle of current software development.

What's special about structure?

Before delving deeper into the concept of structure, it is of the utmost importance to introduce the concept of categorical perception and the phonetic phenomenon – audio-acoustic, perceptive and cognitive.

How can an acoustic flux of a physical nature and described by formalisms of spectral analysis type perceptually become the vehicle of a phonological code of linguistic nature ?

Hjelmslev's linguistic theory distinguishes two plans for the language: the content plan and the expression plan and four stratas: the substance of the content and the form of the content and the substance of the expression and the form of the expression. The audio-acoustic substrate of the phonetic sounds characterizes the susbstance of the expression and the phonological values, the form of the expression. The allophones of a phoneme are the substancial unities, while the phonemes of the phonological paradigms of a language are to the contrary abstract, distinctive unities, of a functional essence and subjected to phonological laws manifesting a stratification of their paradigms.

How can we understand the form of expression which is an abstract morphology of articulation categorizing the continuous phonetic substrate in discrete phonological unities, can emerge while structure from the organization of the susbstance of expression? We chose to introduce this phenomenon because it is part of a vast class of analogous problems encompassing morphogenesis in biology, Gestalttheory in perception psychology, semiolinguistic structures as well as the architectural and urban morphogenesis. It may work as a link between phonetic substance and phonological form, throwing light on categorical perception phenomena. Intuitively anyone can understand it.

The fondamental phenomenon of categorical perception enables us to understand how perception can spontaneously discretize the audio-acoustic flux or how can the discontinous emerge from continuous? Its interest here is rather as a metaphor to understand similar phenomena. The phonemes codified in the audioacoustic flux are immediately categorical to the perception; they have a psychological reality while discrete unities.

The categorical perception phenomena may be likened to critical phenomena analogous to those occurring in thermodynamics characterized as phase transitions.

To understand the categorical content implies in categorical oppositions, so original as those concerned with continous/discrete or quantitative/qualitative. Its objectivity is bimodal, belonging both to a regional physical ontology and to a regional structural ontology.The opposition between continous/discrete – indeed continuous/discontinuous/discrete is fundamental in mathematics. The opposition quantitative/qualitative has undergone a true sense mutation. It is preposterous to say that the qualitative is the impoverished quantitative.

Indeed to talk about structure is talk about the qualitative emergence out of quantitative, emergence absolutely linked to that of the discontinous out of continuous, and more precisely to the development of mathematical concepts such as those of singularity, deployment, bifurcation and stratification.

Indeed the structure concept is a categorical concept shaping the sense of being and the objective content of some phenomena. **Structuralism insists Petitot-Cocorda begins when one faces ideal, non-material structures, abstract forms of organization that cannot be reduced to systems of components in interaction.**

The organizational concept of structure is an authentic comprehensive and explanatory category. Thanks to this rational attitude, theory was introduced in biological and human sciences. Its horizon describes phenomenologically (in Husserl's sense) the formal relationships of dependance linking organically all parts of a whole – parts that mirror forms and relations of a global organization, unable to be assimilated to individual components. They reject a sequential order and call for higher levels of order such as generative and implicate order [Lour98].

While ideal form of organization of a substance, a structure is not a sensitive phenomena. It is invisible however its substancial accomplishment and its effects are observable and can be the object of experimental well defined protocols. In this sense, all structure is a theoretical object (and not a fact), ideal and real. Gilles Deleuze insists: pure "virtuality of coexistence which preexists to the beings", a structure incarnates in its substrate and expresses itself through it. The structures cannot be perceived or observed; they are real while demonstrable. Petitot-Cocorda defines structure as follows: sets of relational and interdependent relationships, the description of their reality is given by a theory and are realized by a visible or observable object which condition its stability and intelligibility.

Umberto Eco reminds us of its ambiguous nature: Is the structure an object while it is structured or the set of relations that structure the object but one can abstract from the object? The structure cannot be detached from the substance it incarnates or the substrate where it becomes substance. It is both intelligible skeleton and structured object. Theorization of the phenomena demands it must be conform to the thing itself. Hence if we care about the structural concept while objective concept or while experience category, we can consider an ontological conception of structures (realist). In an epistemological conception it is reduced to an operational concept of metalinguistic nature whose reality is not ontological but purely methodological.

In the realistic perspective its is a concept that despite its empirical validity acquires an ontological content, an objective value and a constitutive bearing. However it does not subsume the sensitive phenomena and a new class of phenomena non-sensitive must be rationally founded where it can be accomplished and its ontological content determined. It demands to be in accord with the thing itself, the substitution of its semanticism for a mathematical content explicitly built.

II.1.1 The relevance of the categorical perception phenomenon

While conceptual system, methodology and regional ontology (Husserl's sens), the structuralism is essentially transdisciplinary. In all domains Piaget stated it covers a positive common ideal of intelligibility, while epigenetic and relational doctrine of organization, it is next to Physics the only domain where the rational unity of very different phenomena becomes reality. It is fundamental to understand structures while phenomena.

Notions such as continuous/discontinuous/discrete, relation, difference, opposition, junction, transformation, operation, etc are primitive concepts, hence undefinable, belonging to the status of **regional categories.**

If one manages to endow them of a formal expression, one can axiomatize the descriptive metalanguage that is theory and to convert it to a formal language, a pure algebra.

The structure category is always the same and its categories own a content which at last analysis calls for a topological intuition (position, junction, paradigmatic categorization, connection, etc). All structure is above all a stable system of connections between positional values and only exists while such. All structure is a combinatory with formal elements which by themselves have neither form, meaning, representation, content, given empirical reality, hypothetical functional model nor intelligibility behind appearances. Its elements have neither extrinsic designation, nor intrinsic signification. They have just a sense: a sense that is necessarily and uniquely of position. This is why the scientific ambition of structuralism is not quantitative, but topological and relational.

The schematicity of structural categories depends entirely on the possibility of determining mathematically *the positional intuition* which plays the role of "form of intuition" for the structural phenomena. It depends on the elaboration of a geometry of position.

When Buffon referred to embryogenesis, he lamented the absence of a geometry of position: *All that is related to a position is not encountered in our mathematical sciences.* Leibniz called *Analysis situs* an art that was not born at his time and which would make us the relation of position among things, it would be also useful and perhaps more necessary to the natural sciences than the art which has only the measure of things as goal; because one needs more often to know form than matter.

The structuralism depends on the elaboration of a general mathematical theory of morphogenesis.

II.1.1.1 Object identity and position identity. (the subgroup relationships in the crystallographic groups of the plane)

The conflict between identity and difference is at the heart of the structural problem. An identity position of a system are determined by a morphology of interfaces in a control space. An identity position is irreducible to an objectal unity and this not by chance but by essence. However there may be cases where one can associate to a position identity an objectal unity to which one can apply the atomist paradigm about the combination of elementary components [Coco85].

Until now I have posed the problem that a structure is a stable system of connections between positional values and exists only while such. So if we do not manage to crystallize the "position identity" we cannot really advance research especially in the morphodynamic

level. Being concerned with my research to generate sustainable cities, I will illustrate one case with the transition from one crystallographic parent-group to its subgroup derived by the subgroup relationships in crystallographic groups (figure)

During the discussions of my interdisciplinary aspects between Mathematics and Architecture with Professor Norai Romeu Rocco, Director of the Mathematics Institute from the University of Brasilia, the fact that it was impossible to "freeze" the exact transition from one crystallographic subgroup embedded in a parent crystallographic group to another crystallographic group puzzled us. The more we try to describe it, the more we realized it was necessary to start the poster from scratch. Then at the exact moment of designing the targeted transition we would realize it would have an instanteous reality as a true "entity". Since then our intention was to freeze this "entity". What does this entail? Apparently, the closest and simplest approximation to tackle this puzzle reminds me of how Peirce develops the argument that mind in a wider sense is localized not only in the brain of a writer but also the materiality of his semiotic medium, namely ink:

A psychologist cuts out a lobe of my brain [...] and then, when I find I cannot express myself, he says, ***You see your faculty of language was localized in that lobe.****No doubt it was; if he had filched my inkstand, I should not have been able to continue my discussion until I had got another. Yea, the very thoughts would not come to me. So my faculty of discussion is equally localized in my inkstand. It is localization in a sense in which a thing may be in two places at once.* [Nöth02]

Winfried Nöth elaborates on this enigmatic quote of 1902 to introduce the concept of *quasi-mind* in order to distinguish between mind in the sense of cognitive psychology and in the processes of semiosis associated with signs "in a very wide sense".

Obviously one should look for mind in the quote above in two places at once. *In the case of the writing author, one place is his brain, the internal locus of sign production, the other is the inkstand, the locus of the external materialization of the sign. Both loci represent two*

aspects of semiosis inseparably welded like the two sides of a coin. It provides two keys to the understanding of the enigma of the mind in the inkstand: the theory of the unity of the sign with its external representation and the theory of the unity of thought and action [Nöth02].

In our case neither the colourful pens available nor the graphical software such as Coreldraw enable us easily to represent the flow of graphical thought. Its fluidity does not allow repeatability. So I suggested if we managed to measure the sides of the fundamental region instead of creating them through Moser's methodology that only allows the derivation of one group to the other organically once the shape of the fundamental region is already given (in the Self program generated, we built a graphical editor that allows us to mimic the most important step in the architectural design reasoning that is the free-hand sketch).

I am going to continue reproducing here Winfried Nöth's analysis for the reader to feel there is *quasi-mind* not only in the brain but also in the machine (pen or software program). It seems thought cannot precede its representation, but comes into semiotic existence simultaneously with it, it is meaningless to search for thought and meaning in the black box of the brain only. Obviously the signs that result from the cerebral activity play a great role insofar as Peirce concluded: *it is much more true that the thoughts of a living writer are in any printed copy of his book than that they are in his brain.* He enhances this idea: *It is wrong to say that a good language is important to good thought merely; for it is the essence of it.*

Consequently, Norai and I became aware that the only way was to appeal to the unfolding of more expressiveness in computer science to tackle this problem. This is the goal of the next part.

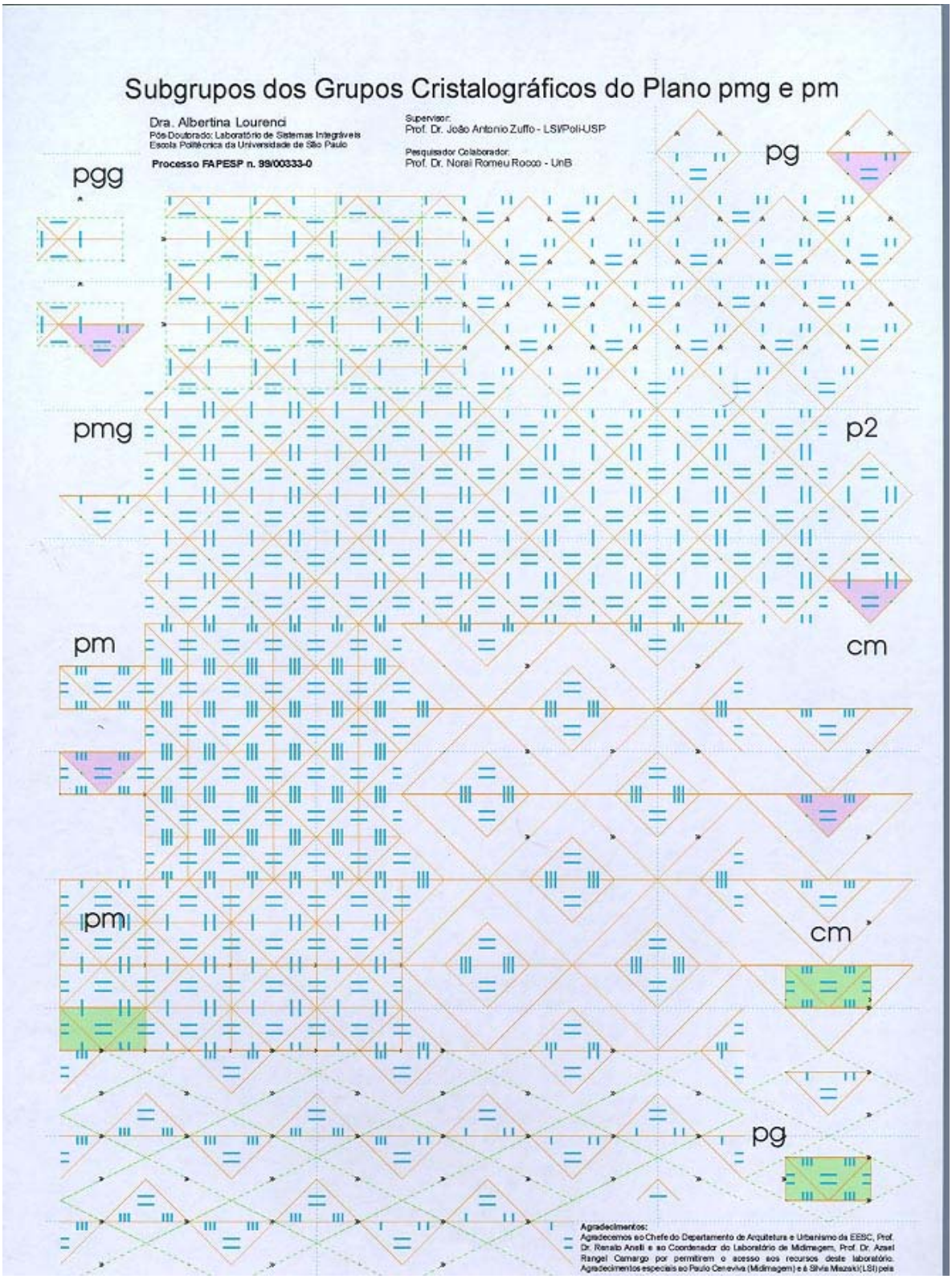

Subgrupos dos Grupos Cristalográficos do Plano pmg e pm
Dra. Albertina Lourenci
Pós-Doutorado: Laboratório de Sistemas Integráveis
Escola Politécnica da Universidade de São Paulo
Processo FAPESP n. 99/00333-0
Supervisor:
Prof. Dr. João Antonio Zuffo - LSI/Poli-USP
Pesquisador Colaborador:
Prof. Dr. Norai Romeu Rocco - UnB
pgg
pg
pmg
p2
pm
cm
pm
cm
pg
Agradecimentos:
Agradecemos ao Chefe do Departamento de Arquitetura e Urbanismo da EESC, Prof. Dr. Renato Anelli e ao Coordenador do Laboratório de Midimagem, Prof. Dr. Azael Rangel Camargo por permitirem o acesso aos recursos deste laboratório.
Agradecimentos especiais ao Paulo Cerevisa (Midimagem) e à Silvia Miazaki(LSI) pela

Figure 3

III. ASPECT-ORIENTED PROGRAMMING

Modularisation as a means to achieve separation of concerns is a key concept in the development of complex software. A fundamental goal is the independent specification of systems aspects and their subsequent integration.

There have been several novel approaches which try to find new dimensions for separation of concerns beyond the "traditional" concepts of module and class. For examples of such "advanced separation of concerns" we cite Adaptive Programming, Aspect-Oriented Programming, Composition Filters, Hyperspaces, role modeling, Subject-Oriented Programming, and so on.

Recently such approaches have been subsumed under the umbrella term Aspect-Oriented Software Development **(AOSD),** whose themes, in addition to programming language extensions, also encompass enhancements of traditional analysis and design methodologies. Such a new and emerging field has yet to garner a uniform conception of its scope and a definition of its central terms. [15]

---

[15] The reader may be feeling uneasy with this general introduction and may be finding difficult to link it to what was stated before. I will quote here again Manny Lehman:

. *Meanwhile let me just add that I have for many years regarded our studies of software evolution as a special case of more general systems evolution, as the fruit fly - drosophila melanogaster - of general systems evolution., the fruit fly because its being non-physical, its rate of evolution is so much higher than that of a biological/botannical system, a vehicle, a weapons system or a city for example. Thus I would expect many of the high level observations to apply although the detailed patterns and rate of evolution will be quite different. Thus the E-type concept applies to all such systems in the sense that, obviously, they evolve and their evolution is strongly driven and constrained by feedback effects and by real world interactions. In this wider sense, too, Systems based on OO concepts, approaches and technology are also of type E (even if their components are type S, ie formally and "completely" specified)* [Lehm01

It is not an easy task to try to make things clearer because this is not the way the computer experts address the problems. At ECOOP'01, which is the most renowned conference in Europe for the Object Oriented community, I had the opportunity to ask **Harold Ossher**, one of the creators of the approach called *Multi-dimensional separation of concerns and the Hyperspace Approach (HyperJ)*, since I am an architect if was indeed necessary to deal with these topics focusing on implementation code. A strong criticism is being addressed on this because all their technological innovations happen at compile time or runtime. For me they should happen at domain model and then reflected through isomorphic reasoning structures in the software architectural model and in the constructs of the programming language available to the programmers at design level. Something similar to what is offered by Self and its subjective version that tries to cope with the issue

[

At ECOOP'01 in Budapest last June, I attended the tutorial given by Mehmet Aksit and Lodewijk Bergmans about *Advanced Software Composition: Obstacles and approaches* [BA01]

The diagram in Figure 1 shows the history of AOP languages.

---

that is being addressed. I was quite surprised **because he confessed honestly that indeed this would be the right approach but he does not know how to do it.**
Well,the criteria adopted by FAPESP to send people abroad are not concerned with this sort of interaction. FAPESP does not realize that the published papers at the main conference seem they are being born dead! A bright language like Self after wide divulgation in 1995 in all conferences is ignored by the majority of computer science experts because Sun Microsystems Laboratory aborted it to give way to Java, a class-based language.
Why? **Simply because computer scientists do not reason semiotically.** Hence the language has no users. However the lack of expressiveness and the difficulty in maintaining software programs and the high costs of a software development are forcing them to ponder over these difficulties. Following a bottom-up reasoning, the essence of experience, with great difficulties they are abandoning inadequate Aristotelian or strict scientific reasoning.
that led to S-types.
Indeed Pattee [Patt96] stresses that physical laws and semiotic reasoning requires disjoint, complementary modes of conceptualization and description. Controls are local and conditional. Life originated with semiotic controls and indeed creation in nature is an eternal hermeneutic game. Art is only its caricatured mirror. Semiotic controls require measurement, memory, selection, interaction, cooperation, none of which are functionally describable by physical laws that, unlike semiotic systems, are based on energy, time and rates of change.
At no hypothesis this means that semiotics is unable of describing evolutive phenomenon. To the contrary: the essence of evolution seen by Peircean eyes resides in the category of infinite and continuity and real generality primarily related to the future in which it acts as a regulative principle of human actions to guarantee the completion of the universe's real lawfulness. Moreover Peirce highlights: *The very being of the General, of Reason, consists in its governing individual events. So, then, the essence of Reason is such that its being never can be completely perfected. It always must be in a state of incipiency, of growth. It is like the character of a man which consists in the ideas that he will conceive and in the efforts that he will make and which only develops as the occasions actually arise…Under this conception the ideal of conduct will be to execute our little function in the operation of the creation by giving a hand toward rendering the world more reasonable whenever, at the slang is, it is "up to us to do so".*
Evolution in semiotics is a blatant difference apparently from the physical reasoning. Indeed I would say they are isomorphic in essence. A physicist tries to unravel the invariants of a complex system little by little. He makes inference inductively. When there is enough evidence, he creates general laws. The problem with semiotics reasoning is the fact that what is being unfolded has an arbitrary and variable nature like the thoughts of a man that will never be the same. Yet the man grows taller if his cognitive processes are given the chance to develop towards more and more encompassing consciousness levels.

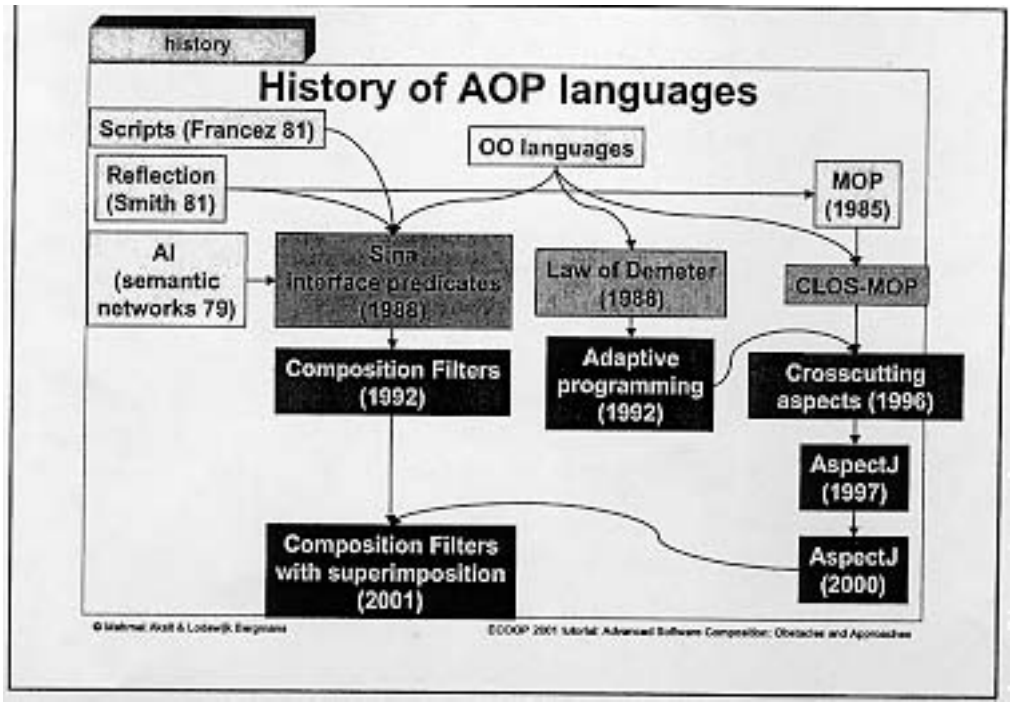


Figure 1.

III.1 Composition filters

Does object-oriented programming really fulfill all the promises? This is the generic way they feel the need that sometimes the programmer experiences composition problems: you cannot reuse or extend the code. Then he gives an example of how to make extensions to Email. He introduces then the notion of multiple views through the example. To be able to make a good implementation of views in an object-oriented system, we must first supply the conceptual model of views. This model captures the different aspects of views. We can make them explicit by making the appropriate abstractions of the state space [Brui98].

So they arrange things within software components. Object interfaces can be divided into several sections, called views to achieve complexity reduction. They call this vertical object division. Horizontal object division is called layering.

Layering is done for several reasons. One of the most important reasons for applying layering techniques is reduction of complexity. Hence large software systems are divided into several layers, which can then be designed, built and maintained more or less independently of each other.

Another reason for using layering techniques is separation of concerns. This separation also enables us to better understand things. One often puts basic functionalities in the core, and then wraps around this core one or more layers with special functionalities. This improves the maintainability of the software system.

Moreover like Günther Kniesel [Knie00], the author of Lava (Java + delegation) Aksit realizes the importance of delegation and tries to mimic it through composition filters. However when they do this, they basically misunderstand the nature of interaction that is a unifying paradigm achieved with great difficulty in prototype-based languages like Self. Interaction is the key factor in the simulation of hermeneutic reasoning. Cooperation is the underlying factor so nicely achieved through a user multiprogrammable virtual reality called Kansas in Self. You can sit at any place in the world and see what your "neighbour" is doing and chat with him/her. All this holistic and artistic, interactive and cooperative reasoning is destroyed in composition filters. Worse all the achievement happens at runtime! This highly centralized control is expressed as a great achievement in figure 2 .and the praised characteristics of the approach are in figure 3 .

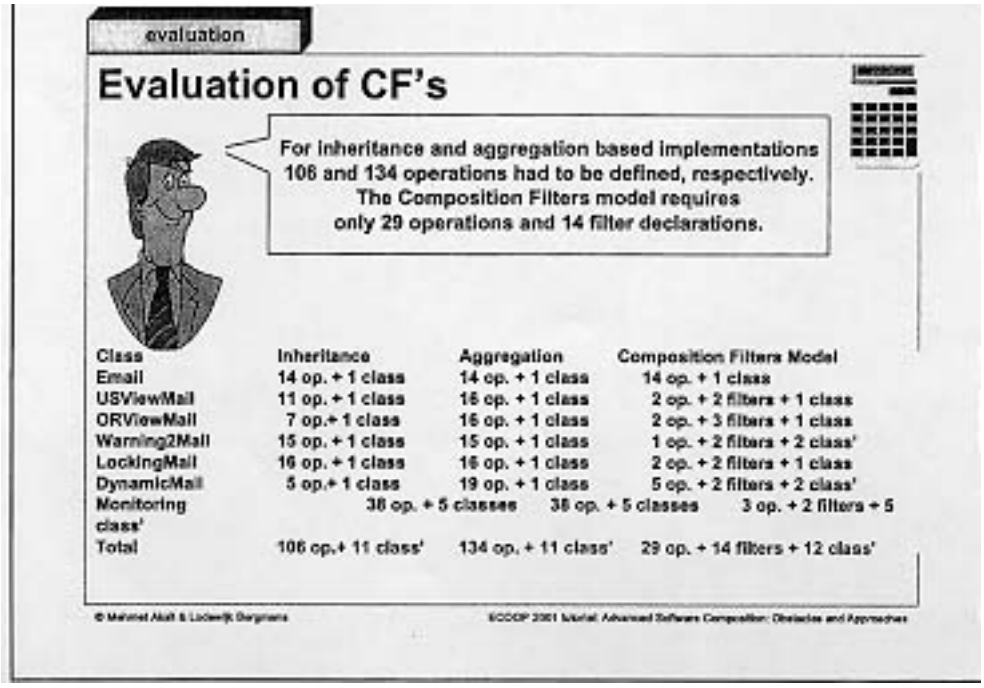


Figure 2. Evaluation of composition filters.

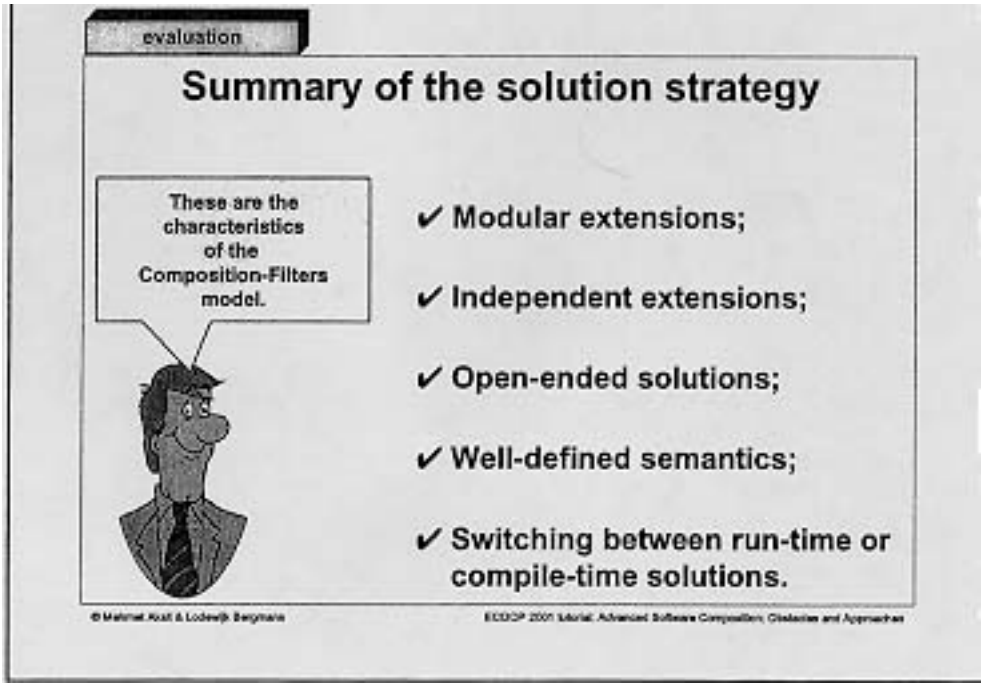


Figure 3. Characteristics of the composition-filters model.

III.2 Multi-dimensional separation of concerns

At the Workshop on OO Architectural Evolution I attended at ECOOP'01, there was a strong criticism of layering. It was suggested this issue should be treated as separation of concerns.

1) The notion of anticipating and designing for the "most likely" kinds of changes to alleviate the impact of future evolution is one of the factors that caused the appearance of the multi-dimensional separation of concerns.
2) According to Ossher and Tarr [OT99], the introducers of subject-oriented programming, it is clearly not possible to anticipate all major evolutionary directions. The semiotic hypothesis allows to build models aligned with evolutionary trends.
3) The "tyranny of the dominant decomposition" as procedures, rules, objects, classes, functions becomes oppressive whenever the concerns a developer has at some point during the lifecycle do not match any of the ones that have been or can be encapsulated. The outcome is scattering, tangling, cascaded, high-impact changes in the code. The software is hard to maintain and evolve. 4) Cross-cutting concerns (better known as aspects) indicate an integration problem among two or more concerns of a design. This in addressed in the composition filters, in Ossher's and Tarr's approach and obviously in aspect oriented programming.

AOP languages provide linguistic mechanisms for:

- Representing crosscutting concerns as separate abstractions
- Specifiying the join points in the source code or in the execution of the program – where the concerns are to be integrated and

- Binding concerns to the join points.

III.3 Aspects

While Mehmet et al and Ossher et al are mainly concerned with Object Oriented programming, Gregor Kiczales et al understand software design processes and programming languages in a mutually supporting relationship. Differently from Jim Coplien and me, they do not take into account the domain model. They stick to the mainstream experts concerned mainly with programming languages, which provide mechanisms that allow the programmer to define abstractions of system sub-units, and then compose those abstractions in different ways to produce the overall system. A design process and a programming language work well together when the programming language provides abstraction and composition mechanisms that cleanly support the kinds of units the design process breaks the system into.

Kiczales realizes that generalized-procedure (**GP**) languages such as OO languages, procedural and functional languages can be seen as having a common root in that their key abstraction and composition mechanisms are all rooted in some form of generalized procedure. These languages break systems down into units of behaviour or function.This style is called functional decomposition. The nature varies according to the language paradigm, where each unit is encapsulated in a procedure/function/object. This unit encapsulated works as a functional unit of the overall system.

However he is specially concerned with programmed properties that must compose differently and yet be coordinated or cross-cut each other. Because GP languages provide

only one composition mechanism, the programmer must do the co-composition manually, leading to complexity and tangling in the code.

Kiczales defines a property that must be implemented using a GP-based language with respect to a system and its implementation as:

A component, if it can be cleanly encapsulated in a generalized procedure (i.e., object, method, procedure, API). By cleanly, he means well localized and easily accessed and composed as necessary. Components tend to be units of the system's functional decomposition such as bank accounts, and so on.

An aspect, if it can not be cleanly encapsulated in a generalized procedure. Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways. Examples of aspects include memory access patterns and synchronization of concurrent objects.

A GP-based implementation of an application consists of:

(i) a language,

(ii) a compiler (or interpreter) for that language,

(iii) a program written in the language that implements the application.

Likewise an AOP-based implementation of an application consists of:

(i.a) a component language with which to program the components

(i.b) one or more aspect languages with which to program the aspects

(II) an aspect weaver for the combined languages

(iii.a) a component program that implements the components using the component language, and

(iii.b) one or more aspect programs that implement the aspects using the aspect languages.

Just as with GP-based languages, AOP languages and weavers can be designed so that weaving work is delayed until runtime (RT weaving) or done at compile-time (CT weaving).[16]

III.3.1 Extending aspects to domain model and software architecture

Many changes to a software system can be made without knowledge of the full details of the system's implementation. It is desirable to permit developers to work at the highest level of abstraction possible and to facilitate navigation between different levels of abstraction as needed, since doing so reduces the complexity of development tasks and promotes comprehension.

The interactions among different components also appear different at different levels of abstractions, just as the components themselves do. Thus, the ability to model and navigate different levels of abstraction effectively depends on the ability to describe both components and their interactions at different levels of abstraction [TDBL00].

The engineering principles modularization and separation of concerns do not always fit well together because a module only represents a coherent whole that encapsulates a certain

[16] **Of course the fact that Self, Smalltalk and LISP and CLOS are based on exploratory programming and hence at each step of the design the implemented code runs, so that when the implementation of the design is over, the implemented code makes reviews unnecessary (at each step if a problem happens, there is a debugger to check what is happening at the local of the bug) is overseen by Kiczales. This is the way computer scientists act. And they call this science!!!Great improvements in programming are simply overlooked!**

well-defined aspect of a system. They do not cope with the need to assemble systems with certain desired properties that cut across the components, so that some of their facets have to be changed to implement a particular desired property.

The introduction of aspects deals with this but they are traditionally defined at the implementation level, and thus have to be based on a very fine grained level of abstraction. In contrast, many higher-level software concepts such as patterns and architectural styles, are only implicitly contained in code.

Without knowledge about this higher level structure it is extremely difficult to change one aspect of a system only, while leaving the other properties intact.. Colin Atkinson and Thomas Kühne show how to do this through Stratified Architectures [AK00].

My ecodesign model entitled *Model of Primary, Secondary and Tertiary Waves to design and plan sustainable cities* also deals with separation of concerns at the domain level. Moreover the underlying geometric modeling cares for composition of concern especially cross-cutting concerns (Part II .IV).

III.4 Reflection and metaobject protocols

Aspect-oriented programming has a deep connection with work in computational reflection and metaobject protocols [Kicz91]. A reflective system provides a base language and (one or more) meta-languages that provide control over the base language's semantics and implementation.

Several approaches can be identified for building reflective systems. They can be classified according to the paradigm that is used for combining object-computation and reflective computation.

First, there are systems that incorporate some form of reflection in an ad hoc way. For example, learning systems necessarily exhibit some sort of reflective behavior since they are able to make improvements to themselves.

In section II.3 Widening the reflective nature of the MPSTW/SGPGM I try to show how reflection capabilities are intrinsic to my ecodesign model and its underlying geometric modeling.

Other group of systems makes use of reflective facilities provided by the programming language in which they are implemented such as LISP and Self.

Jecel de Assumpção recognizes reflection as a fundamental programming concept to create truly open computational systems. Hence he is creating Self/R.(Appendix I) [Assu01].

This implies that the language provides syntactical constructs for specifying reflective code and that the interpreter of the language is able to actually execute this reflective code during computation.

So a language with a reflective architecture supports (besides object-computation) the explicit and uniform representation of the reflective computation of the computational systems that are implemented in it [Maes87]

A programming environment has a **meta-level architecture** if it has an architecture which supports meta-computation, without supporting reflective computation. A meta-level architecture is designed for building systems which are about other systems. However, it does not make it possible to build systems which are about themselves. Of course reflective architectures can be built on top of any GP languages.

The meta languages provide views of the computation that no one base language component could ever see, such as the entire execution stack or all calls to objects of a given class. They cross-cut the base level computation.

In AOP terms, metalanguages are lower-aspect languages whose join points are the "hooks" that the reflective system provides. Indeed the aspect language is implemented as a meta-program, called at each method invocation, which uses the join point information and the aspect program, to know how to appropriately marshal the arguments. Thus the higher level aspect language designed by Kiczales is implemented on top of a lower level one. Curiously all the promising research with reflective architectures that would yield open computational systems has stuck like Agora from Vrie University. Pattie Maes is working with multi-agents. The strong background Kiczales had in reflection seems to be tapped to AOP which he recognizes is a goal for which reflection is one powerful tool. He also considers composition filters as reflective facilities.

III.5 Multi-paradigm design. Conclusions

Jim Coplien shows explicitly in his PhD thesis that it is impossible to model the real world within the perspective of a single paradigm. Moreover he clearly emphasizes the importance of viewing domain model, software architecture and programming language as tightly connected [Copl00].

Greimas's semiotics believes that semiotics is not a theory of the signs, rather it is a theory of meaning that becomes operational when its analysis is elaborated at levels above and below the sign [GC79].

I would say this resonates with the gist of Coplien's PhD thesis.

Likewise it is my intention to develop a knowledge system conceived as a true open meaningful system that takes into account an analysis in the level beyond and below the functional unit of the programming languages. This is indeed a must if I want to develop it in a seamless way. Indeed the domain model was created as an amalgama of different paradigms such as catastrophe theory, Hjelmslev's semiotics, graph theory. Since it has an evolutive nature because it reasons as a semiotic, hermeneutic and autopoietic entity, its evolution unfolds naturally and more encompassing paradigms envelopes it easily, such as Peirce's general theory of the sign. So I need to map this rich structure into the software architecture and the programming language.

I will continue this analysis because **the prototype based object oriented programming language Self is able to be embellished by reflective architectures, aspects, separation of concerns and so on. These are obviously new paradigms in the sense elaborated earlier in this chapter.**

Concerning the high-level mechanisms - those that specify the simultaneous interaction of several objects, classes, a system or a framework -while languages do not typically provide the means for the user to develop higher-level abstractions, design patterns do [GOF [BMRS+].

On the one hand, experts interested in higher level lingual abstraction mechanisms[17]- have done little research to understand and promote the key concepts in component-oriented programming; that is, identifying what exactly is component-oriented programming and what language mechanisms exist for component-oriented style of programming and how to express these key ingredients in a component-oriented programming language.[SD01].

I have reviewed a great number of papers about this, but they are not conclusive.

Nobody can deny the importance of the introduction of the design and architectural patterns. However they present lots of problems {Bosc]. And alternative solutions are being outlined to make their use less clumsy. Definitely if they were implemented in prototype-based object oriented languages many would disappear and their implementation would become easier**. Why instead of promoting delegation based language, the OO community insists on simulating it however sticking to class-based languages?**

Indeed they sort of depend on the development of lower-level mechanisms.

On the other hand, intensive research concerned with lower level mechanisms are controversial. I will briefly try to outline this. Of course the whole section is concerned with this topic.

Concerning the lower-level mechanisms such as intraviews(perspectives) and intralayers[18], the subjective version of Self, called Us [SU96] has tried to advance research in this direction.

[17] many models for component-based software development, are based on sets of standards and frameworks (APIs)), and are implemented on top of a mainstream object-oriented programming language

[18] A multi-layered-view approach entails intra-, inter (cross) - e super views and layers [MG01].

Procedure invocation in a simple procedural language (Fortran, say) can be thought of as a kind of message passing in a degenerate case in which there is only one message receiver. The message name is looked up in a kind of giant virtual dictionary, and a resulting "method"(procedure) is invoked. For example the Fortran expression sqrt (2.0) sends the sqrt message to the (implicit) world, with argument 2.0. Since there isonly one receiver (the entire world), the sqrt message always runs the same code (square root routine).

Generalizing from procedure- to object-oriented programming imposes another indirection in method lookup: in OO programs, there exists more than one potential receiver (just as in 2-space there is more than one possible y coordinate) , so one must send a message to **some object** by by specifying the receiver. Each object needs its own virtual dictionary of message-to-method mappings. All objects that send messageX to object Y will get the same result.

From the viewpoint of design patterns, we would include **inheritance**, **encapsulation** and **polymorphism** as **patterns** to procedural languages [GL98].

A world populated by objects is richer than the simple procedural system.It allows you to see the world as it is, not as atoms or molecules as in the case of procedural programming.

Kant emphasized the importance of direct experience of things in themselves: *The realist will hold that the very same objects which are immediately present in our minds in experience really exist just as they are experienced out of the mind; that is he will maintain a doctrine of immediate perception. He will not, therefore, sunder existence out of the mind and being in the mind as two wholly improportionable modes. When a thing is in such relation to the individual mind that that mind cognizes it, it is in the mind; and its being so in the mind will not in the least diminish its external existence. For he does not think of the mind as receptacle, which if a thing is in, it ceases to be out of* [Apel81:31-32]

However it is still an "objective" world in that a reference to an object gives access to a particular set of behaviors, regardless of the reference holder. It does not allow the perception that an object according to Peirce is a successive multitude of inner and outer parts. Then to reach full subjectivity and engage an hermeneutic game in software development, Randall Smith and David Ungar propose another level of indirection be applied to message lookup: somehow, a perspective or "point of view"that can be specified by the message sender should be a participant in the lookup algorithm. Before a message can be sent to an object, the system must consider the perspective from which the message is sent, just as in a 3-world, a third coordinate must be taken into account when locating points.

Then full subjectivity is reached in agree with the concept of unlimited semiosis : this ideas happens in the form of a dialogue and may be likened to the hermeneutic circularity in the dialogical process between me and the other [Nöth98]. X-programming and agile methodologies are trying to mimic these cognitive processes in software development. It is much more efficient if the programming language also mimics the human cognitive processes such as these. When Alistair Cockburn emphasizes people as first-order linear components in software development as well as agile methodologies, he resonates with Peirce: *It is plain that this view of reality [namely, that one which defines the "external" reality of things, insofar as it is independent of actual opinions about it, by means* ***of its cognizability in the ideal final opinion of the unlimited community of researchers****] is inevitably realistic [regarding the theory of universals]; because general conceptions enter into all judgements, and therefore into true opinions…It is perfectly true that all white things share whiteness in them, for that is only saying, in another form of words, that al white things are white; but since it is true that real things possess whiteness, whiteness is*

real. It is a real which only exists by virtue of the act of thought knowing it, but that thought is not an arbitrary or accidental one dependent on any idiosyncrasies, but one which will hold in the final opinion [Apel81:30].

Indeed prototype based languages like Self allows the inspection of the thing as it is. Us would allow us to dialogue about the thing and uncover its myriad of aspects. X-programming speeds up this awareness, introducing explicitly the opinion of each stakeholder in the software development.

Us tries to achieve this at the design level (this is superb) and treats each perspective as an object. Central to their formulation of perspective, different views about the same object, is the concept of a layer. As I have already pointed out, one needs to sharpen terminology here before sharpening the tools [Mitt01].

A layer may have another layer as a layer parent, and the layers therefore form a hierarchy. A layer considered together with its layer parents is a perspective. So again the idea of hierarchy creates restrictions that may hinder "a free dialogue". So it is necessary here to ponder over if it is not preferable to introduce the separation of concerns and aspects approach in Self/Us and build a seamless programming language.

But what is fundamental here for our argument is the idea that:

Each object has exactly one piece on each layer.

A piece is a collection of attributes (variables and/or methods) and is properly considered that part of an object which is associated with a particular layer.By bending the layers thought of as sheets of glass into cylindrical sections, we can visualize all the pieces for a single object filling a horizontal disk-shaped region. An object reference can then be thought of as a reference to this disk, the collection of all pieces for the object. This resulting packet is an object as viewed from a particular perspective. The introduction of

subjectivity in Self might have a high impact in situations such as encapsulation,encapsulation and security, multiple views (MVC), reflection, debugging the user interface, etc!

Although this is a very rich field, there are no solutions up to the present. Higher and lower level mechanisms (in relationship to the object, class functional unit in OO paradigm) are costly. And curiously powerful reflective abilities are necessarily to unfold these mechanisms. Reciprocally they also enhance reflective capabilities.

In chapter V, I will adapt this approach within the context of my knowledge system to generate sustainable cities explicitly as future research trends.

PART II

IV. THE DOMAIN MODEL: *THE MODEL OF PRIMARY, SECONDARY, TERTIARY AND N-ARY WAVES TO DESIGN AND PLAN SUSTAINABLE CITIES* (MPSTW) AND ITS UNDERLYING *SYMMETRY GROUPS OF THE PLANE BASED GEOMETRIC MODEL* (SGPGM)

It was shown in chapter II how complex it is to have a really structured model, if one wants it to be a confirmed simulation of the target domain in the real world (Figure I.2).

Figure 1 shows a multi-paradigm (catastrophe theory + Hjelmslev's semiotics + graph theory) based domain model entitled *The Model of Primary, Secondary and Tertiary Waves to design and plan sustainable cities* . It is an application of the catastrophe theory to developmental biology developed by Zeeman. I extended the secondary waves of his model entitled *The Model of the Primary and Secondary Waves in developmental biology* through application of Hjelmslev's language theory orbiting around the planes of the language and its stratas substance and form. I also had to apply graph theory to transform the phenomena into experience objects [Lour88].

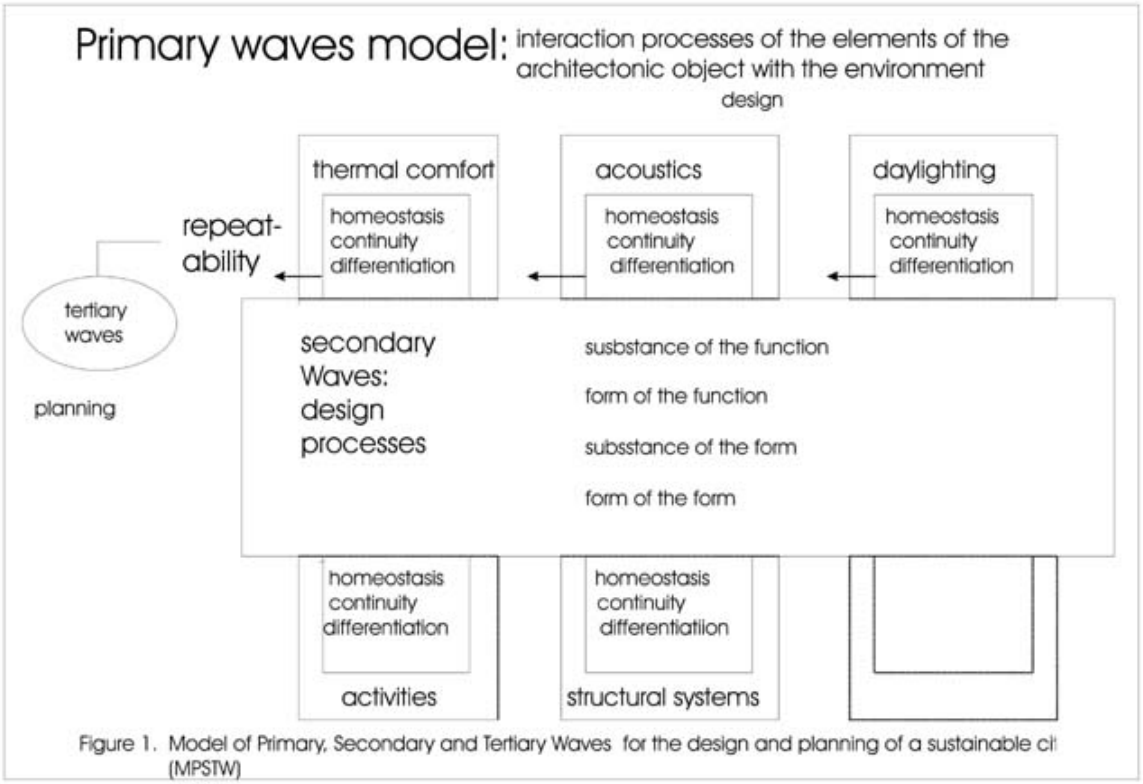


Figure 1. Model of Primary, Secondary and Tertiary Waves for the design and planning of a sustainable ci (MPSTW)

On the one hand, its regional categories subsume the phenomenal diversity involved in architecture, urban design and planning. However they are meaningless if not associated with the target phenomenal diversity. They describe separating all concerns involved in the act of design and planning neatly. There is a regional category for every dimension ranging from concerns with the ecology of the behaviour of the human being, the "thingness" of the architectonic object revealed through its elements , the characterization of the environment (earth, climate, vegetation) to the topological/geometric processes involved in the act of design.

On the other hand, a new geometric consciousness emerge from this substrate, mathematically translating into meaningful form (schematicity) the underlying phenomenal biodiversity of the target sustainable cities. The corresponding **Symmetry Group of the Plane Geometric Model (SGPGM)** composes each concern horizontally and vertically

shaping a final integrated sustainable architectonic object (Figure 2). I will not delve deeper into details because it is out of the scope of this Report but the intrigued reader may look for the master dissertation [Lour88] and the PHD thesis [Lour98].

In the paper entitled *An evolutive architecture reasons as a semiotic, hermeneutic and autopoietic entity* [Lour01b], I praise that this structuring leads to a model fond of change, evolution, cooperation, interaction, promoting infinite synergies. Any change or incremental need fits nicely into any level of the knowledge system not only due to its isomorphic nature pervading all levels but also due to autonomous organization of each level. I mean to boost a component of a level does not necessarily mean to propagate the change to all levels due to autonomy. However if it is decided the change will be stably integrated to the system, isomorphic structures pervading all levels ease this task superbly. Due to the possibility of independently experience with each regional category or process, one can design and implement it interactively, cooperatively or independently. If it passes the test, it may or may not be integrated to the whole model. Yet the model does not have the nature of interactive components or a list of elements because it has a geometric-topological nature and all components are linked to each other recursively or intertwined.

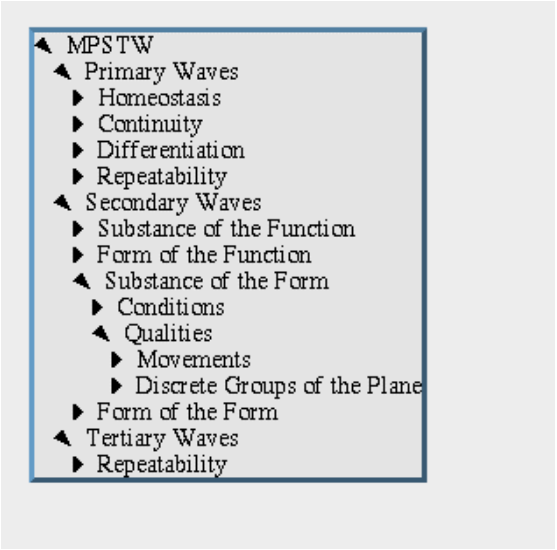
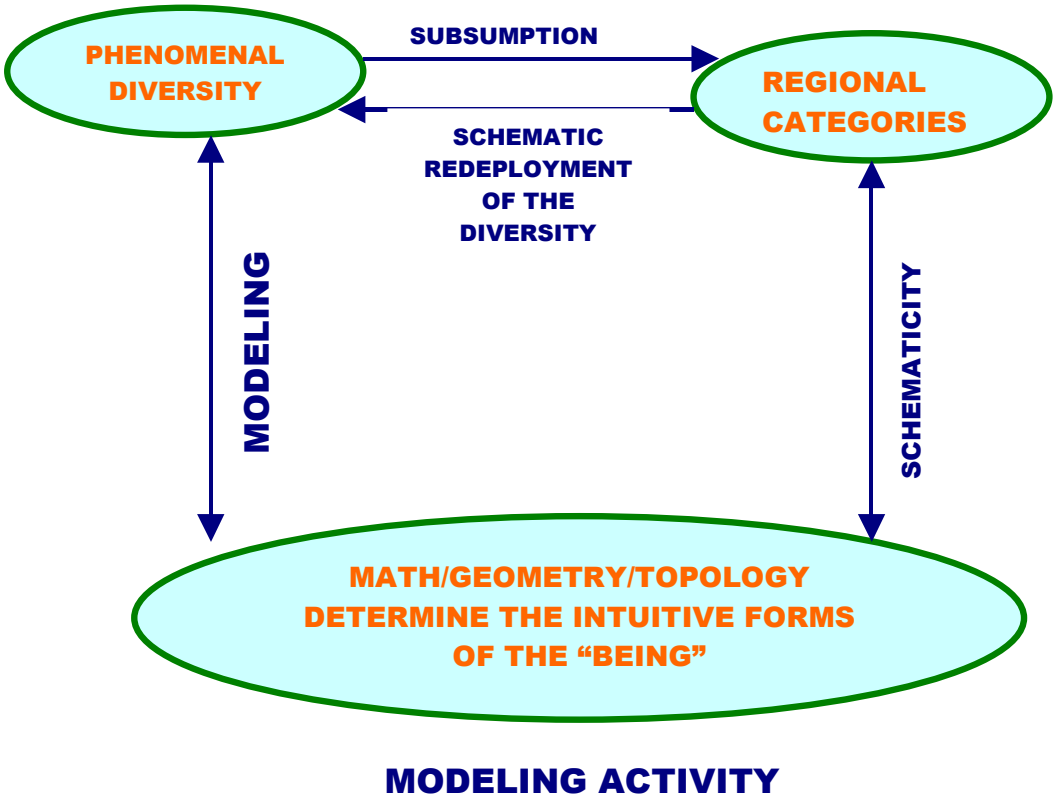


Figure 2 a). A new geometric consciousness emerge from this substrate, mathematically translating into meaningful form (schematicity) the underlying phenomenal biodiversity of the target sustainable cities.

In the paper entitled *An evolutive architecture reasons as a semiotic, hermeneutic and autopoietic entity* [Lour01b], I praise that this structuring leads to a model fond of change, evolution, cooperation, interaction, promoting infinite synergies. Any change or incremental need fits nicely into any level of the knowledge system not only due to its isomorphic nature pervading all levels but also due to autonomous organization of each level. I mean to boost a component of a level does not necessarily mean to propagate the change to all levels due to autonomy. However if it is decided the change will be stably integrated to the system, isomorphic structures pervading all levels ease this task superbly. Due to the possibility of independently experience with each regional category or process, one can design and implement it interactively, cooperatively or independently. If it passes the test, it may or may not be integrated to the whole model. Yet the model does not have



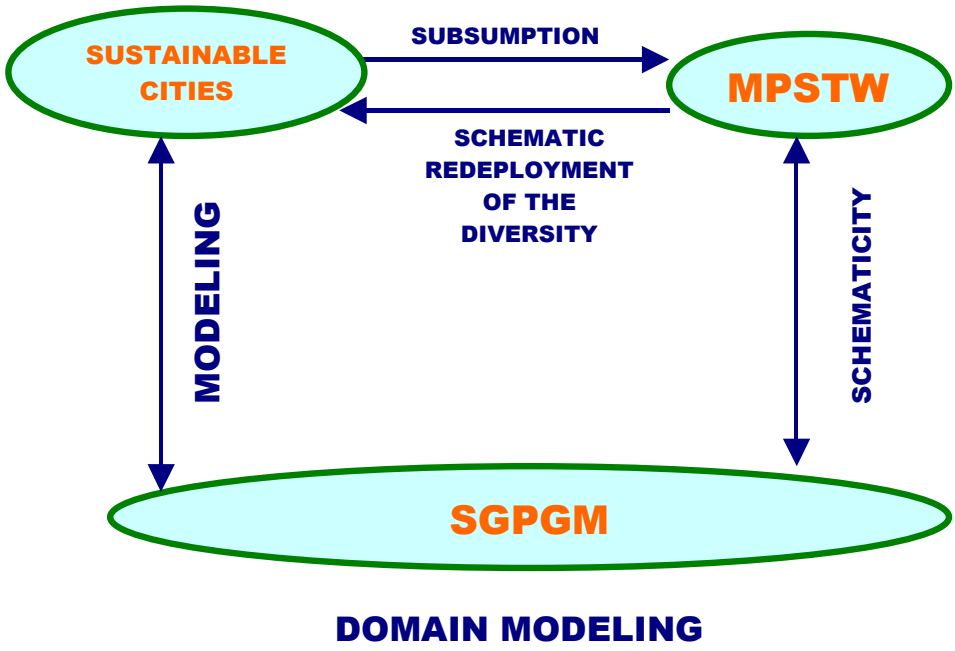


Figure 2b) The corresponding **(SGPGM)** composes each concern horizontally and vertically shaping a final integrated sustainable architectonic object .

the nature of interactive components or a list of elements because it has a geometric-topological nature and all components are linked to each other recursively or intertwined.

II.1 EVOLUTIONARY CHANGES TO THE DOMAIN MODEL

From the referenced reviewed bibliography about the current practice of sustainable projects (buildings and cities), I examined thoroughly case by case the books *Sustainable Architecture* Towards a diverse built environment from Ed Melet [Mele99] and *Ecourbanism. Sustainable human settlements: 60 case studies* from Miguel Ruano {Ruan99]. The former is an initiative taken by the Dutch Government.

All over the world, ecocities have been built especially concerned with **electrical energy generating systems** (photovoltaic systems, wind turbines, biomass, biodigestors, etc); **integrated biosystems** not only in farm but also in cities tuned to zero emissions policy to the environment; permaculture gardens; landscape and agriculture integrated practice at home; **ecohydraulic systems** (water conservation through compost toilets: the compost chamber converts the waste into humus, used in the home garden); rainwater stored in artificial lakes in the roof or in the ground; recycling of grey water in artificial or natural wetlands to remove impurities before reuse for gardening and irrigation; **intelligent electrical systems** (sensor systems for each house, transmitting data to a central computer for performance evaluation and feedback); computational systems integrated to the home as well as multimedia space for interaction with researchers, customers; also multimedia spaces for leisure: different squares of the world interconnected by multimedia facilities, so anyone strolling can talk to anyone strolling in the other side of the world, etc.

Indeed the omnipresence of the Internet and the World Wide Web via phone lines, cable-TV, power lines, and wireless RF devices has created an inexpensive media for telemonitoring and remotely controlling distributed electronic appliances. [Jahn01].

The most remarkable sustainable buildings are the so-called **smart buildings** and **energy-generating buildings**. Energy-generating buildings are social buildings. They satisfy their own needs, but their energy-efficiency is so high that these needs are in any case reduced to a minimum. If they yield an energy surplus, they can sell it to the grid. Buildings of this kind will make an important contribution to the energy economy of the built environment in the near future. The current Brazilian energy crisis points towards their needs.

Energy-generating buildings are not all different from smart buildings. The shape of energy-generating buildings will similarly be significantly determined by local conditions, such as the predominant wind direction or the solar inclination ("form follows climate"). The main difference is that the smart buildings (figures 3, 4) aim to employ natural principles, whereas energy-generating buildings place their fate in the hands of technology. (figures 5, 6).

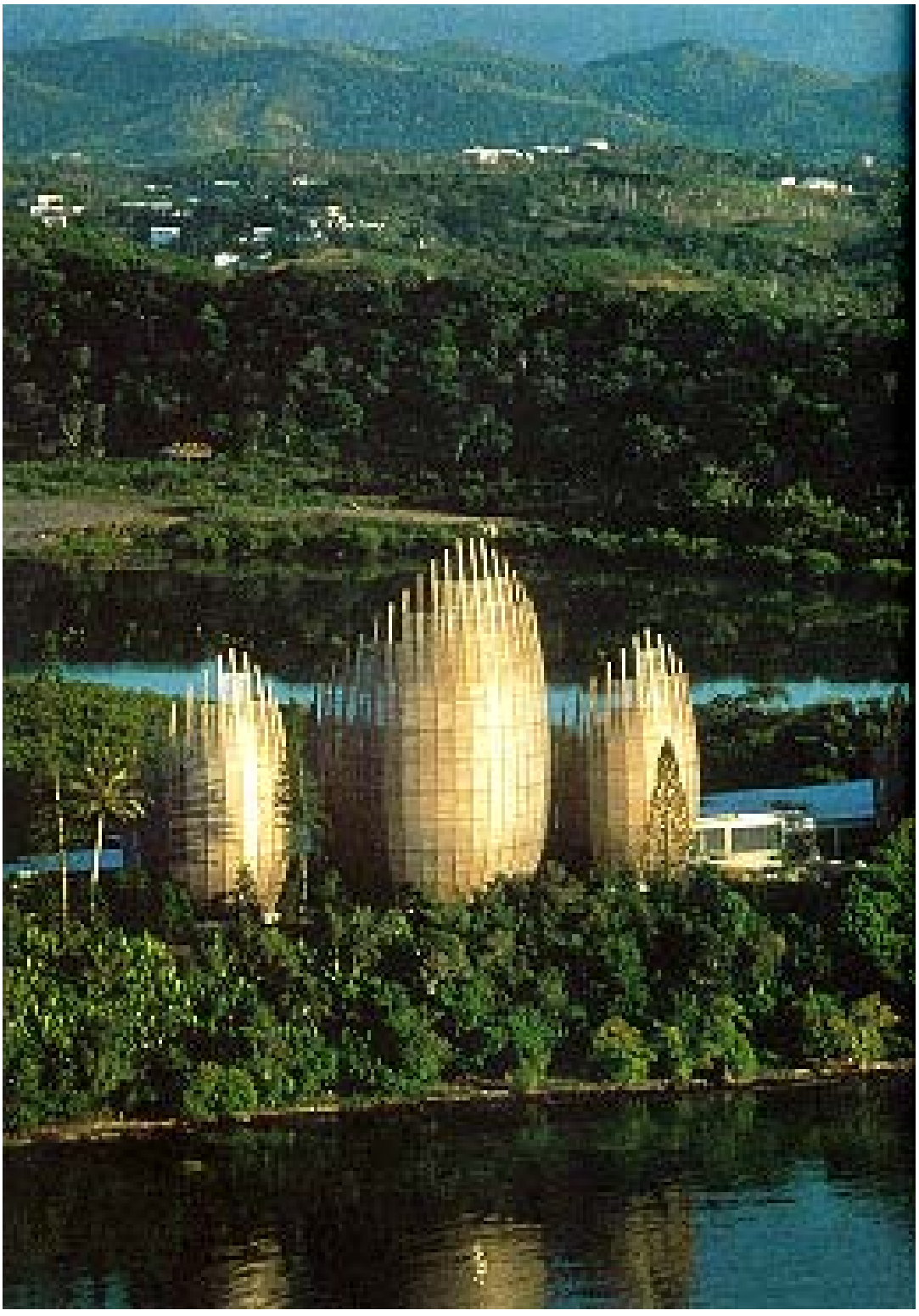


Figure 3. Jean Marie Tjibaou Cultural Centre, New Caledonia. Renzo Piano Building Workshop. Renzo Piano found inspiration for the seaward aspect in the traditional huts of the New Caledonians, which are made from several layers of leaves. Piano used curved, laminated strips of iroko-wood, which block solar radiation and act as a windbreak. Source: [Mele99]

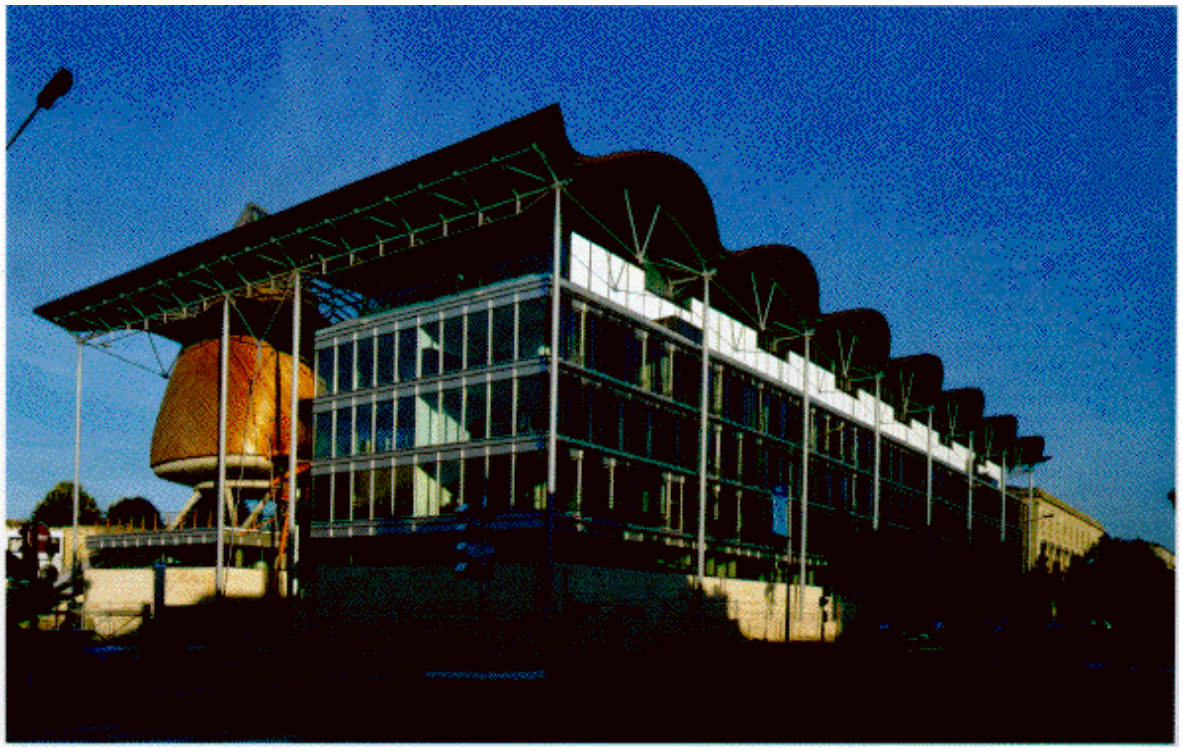
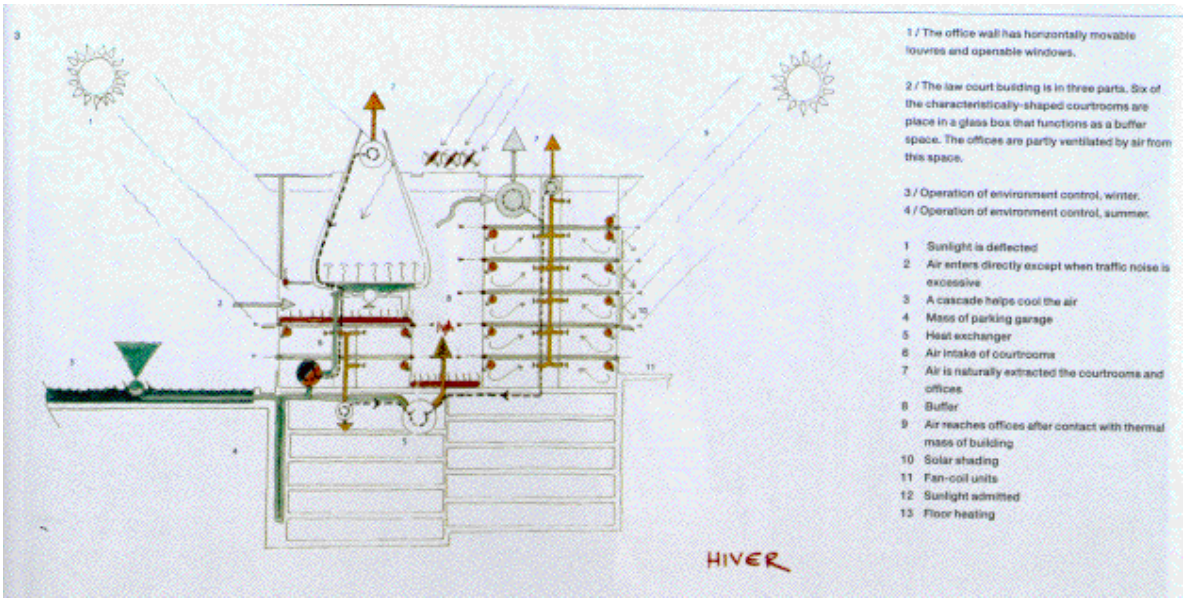
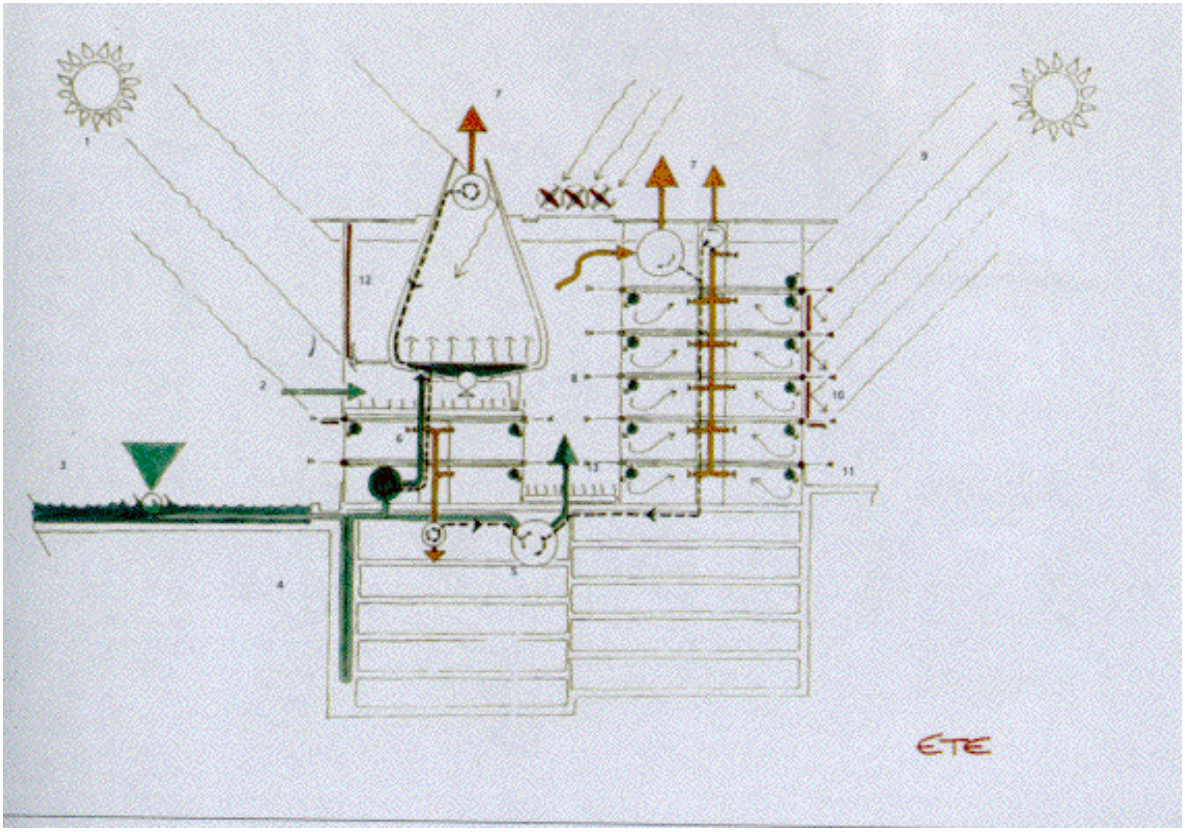


Figure 4. a) Law Courts, Bordeaux. Richard Rogers Partnership. The industry-aspect of the building is highlighted. Six of the seven onion-shaped courtrooms are lodged under the undulating steel roof of the glass buffer. The courtrooms are ventilated by cooled ambient air, which enters them through floor-level grids at extremely low velocity, gradually rises taking much heat with it, and exits through inconspicuous pores in the wooden cladding.





b) The ventilation air is cooled by water flowing down a cascade in the public square and by the mass of the vehicle garage through which the air is conducted. Besides serving the courtrooms, this cooled air is blown into the glazed buffer.Grilles in the façade draw the slightly warmed air form the glazed buffer into the office wing. This air passes through channels cast into the concrete floor and is distributed throughout the offices. Source: [Mele99}

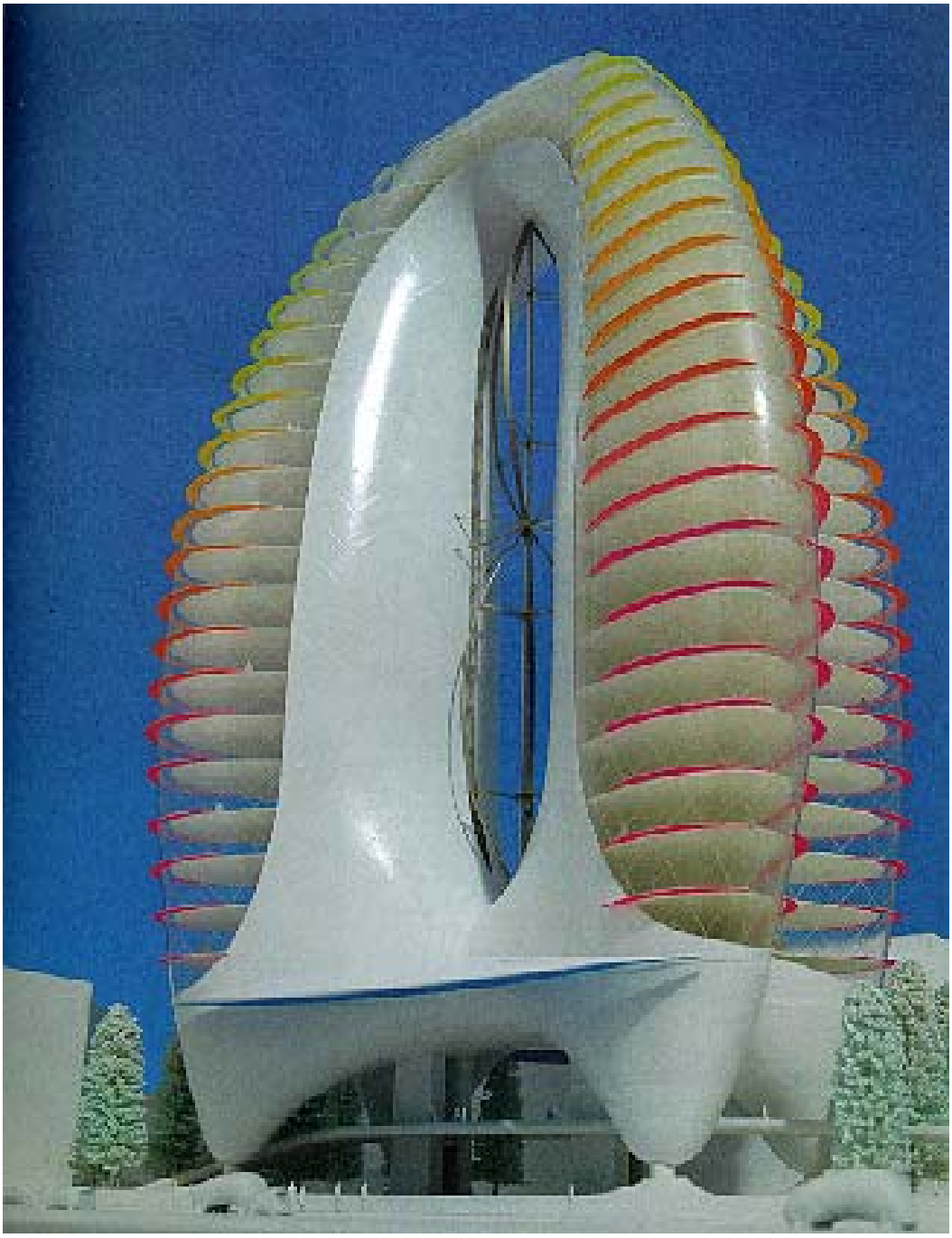


Figure 5. Office building. Project ZED, London. Future Systems. The design of this office building forms part of the Zero Emission Development Project. The orientation of the building ensures it will profit form the prevalent wind, which is conducted through a tunnel in the center of the building. This tunnel is designed to act as a venturi tube, which increases the velocity of the air, considerably raising the yield of the two vetical wind turbines mounted in the tunnel. The main problems of integrating a wind turbine into a building are vibration and noise. To absorb these, Future Systems provided thick concrete walls around the turbines, which not only act as a noise barrier but give the building greater stability. The concrete constitutes a major part of the building's thermalmass.Source: [Mele99]

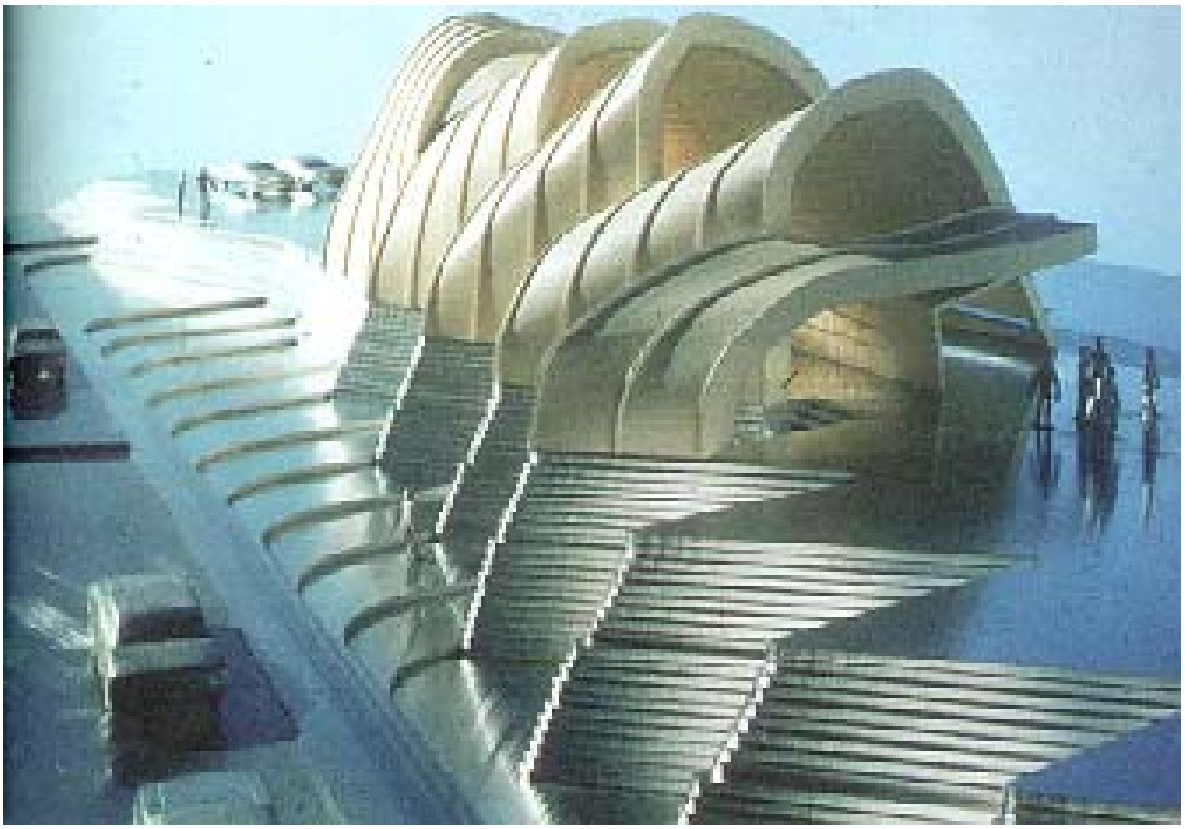


Figure 6. Visitor Centre OVM, Austrian Mineral Oil Company. Schwechat In this pavilion, visitors to the OMV Refinery can learn about new, clean ways of generating electricity. Greg Lynn used state-of-the-art simulation software to model various aspects of the visitor center. Information on the day-to-day motions of the sun was used to optimize the alignment of the photovoltaic cells on the façade. This together with the movements of the roaring traffic on the highway to Bratislava helped determine the final organic shape of the pavillion. The computer model was moreover programmed for the façade to contract when in shadow and expand when illuminated by the sun.The center demonstrates the combined use of photovoltaic cells and fuel cells for the generation and storage of electricity. Source: [Mele99]

IV.1 Evolutionary changes to the domain model

For a non-expert in architecture, the presentation of information above may seem superfluous. However architects and designers firmly believe that their projects "speak by themselves". The

information in the books are not more detailed than the summary above. Yet to examine these huge amount of intriguing cases is a right-sided brain-oriented activity, requiring concentration to acquire information through the careful examination of each plan and section.

Obviously I cannot ignore these breakthroughs. They make evident time is ripe for triggering the Tertiary Waves Model responsible for planning.

With this aim, meetings have been held since last March to consider a possibility of developing a joint thematic project to model an intelligent energy-generating sustainable building. as well as to unfold the Tertiary Waves Model uniting the Department of Energy and Automation under the Prof. Dr. Lineu Belico's coordination and the Laboratory of Integrated Systems under Prof. Dr. João Antonio Zuffo's coordination. It would be held with professors, graduate and undergraduate students from both departments. Several contacts were made with Architectural Schools, for example the Department of Urban Sustainability from the University of Brasilia under Prof. Dr. Martha Romero's direction.

Examining the sustainable projects and cities, it becomes clear the strengths of my research:

1) it orbits around a robust domain model and its underlying geometric and computational models, shaping a knowledge system to design and plan sustainable cities. All around the world, there is no similar concern.
2) its evolutive nature reflected in all levels of the knowledge system may work as an information radiator, influencing straightforwardly the progress of several fields especially if well divulgated through workshops and conferences organized by me

as well as promoted by the Departments involved in the Thematic Project. It may serve as an attraction pole for researchers all over the world.

Although holding the label of sustainable, everybody is aware that the projects being designed were not evaluated in terms of performance and costs to prove their sustainability. And evidently to date they are far from being sustainable. Because they have no know-how to be designed. Only the costs to pay the experts involved make them non-sustainable. So the urge for models that ease the task of designing them efficiently and at costs compatible with sustainable development. Obviously the buildings and cities were designed for wealthy people from the First World countries.

These breakthroughs also influenced the direction of research in terms of its geometric implementation through the symmetry groups of the plane. The breakthrough reported in the last Scientific Report [Lour00] in terms of the transformation of the crystallographic groups into one another through the subgroup relationships (figure 7) enables the designer not only to conform the free plans to one another in any level of the design, namely, the apartment, the building, the neighborhood, the borough, the ecocity, the bio-region, but also to integrate functions vertically. Finally the architect can compose harmony. Before only melody was available.

This has a huge impact on design because now several experts can sit down next to each other or through the multiuser programmable virtual reality infinite screen Kansas available in the prototype-based object oriented programming language Self and simultaneously integrate all processes that characterize the elements of the architectonic object.

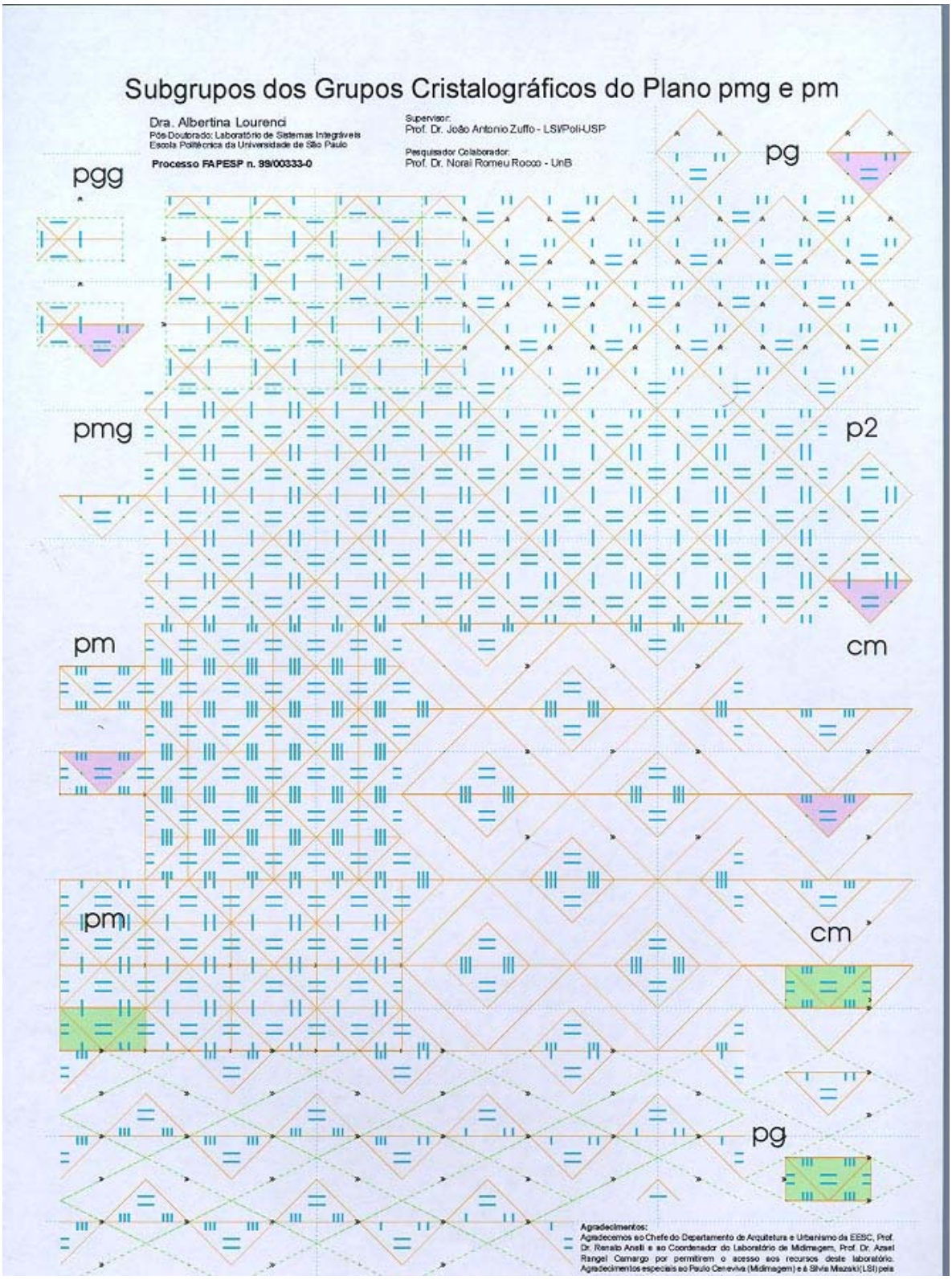


Figure 7. The subgroup relationships of the plane.

IV.2 Zooming in the nature of the categorical perception phenomenon in architectural/urban design

The figures 8. 9 and 10 programmed in Self represent the design processes of the element activities. Their examination begs a question:

How can an undifferentiated thoughtlike flux of a quantum nature and described by formalisms of quantum field type [Lour00:49] **become the vehicle of a morphological code of a geometric nature?**

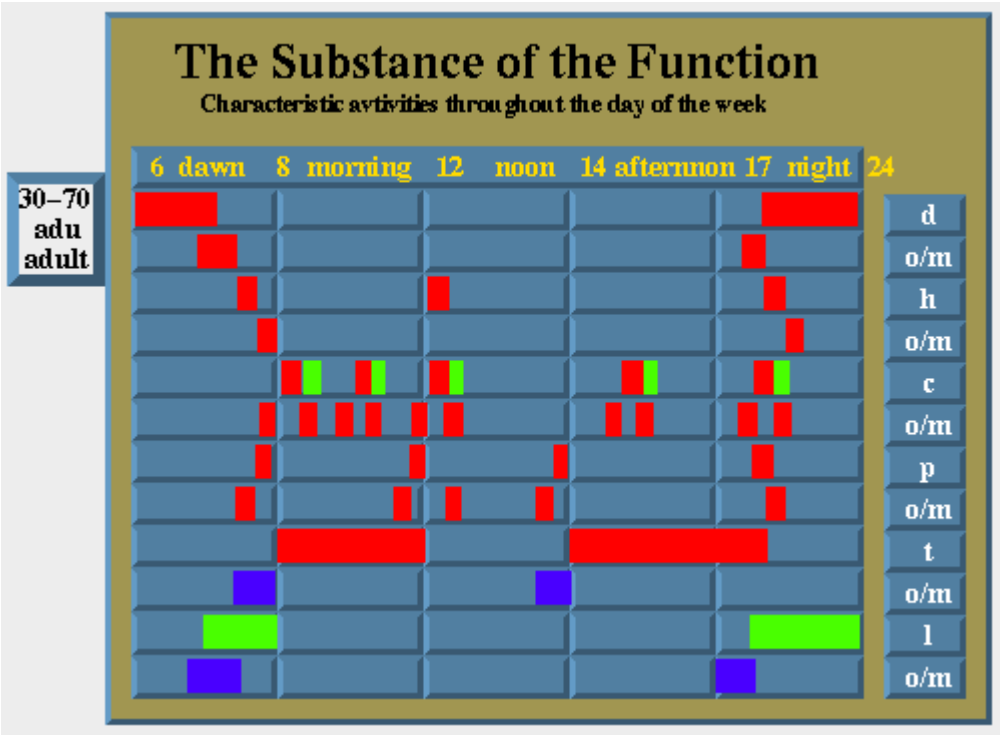


Figure 8. The substance of the function.

Figure 8 describes the characteristic activities throughout the day of the week iterating along the cycle of the human being from baby to old age and taking into account certain special mental states such as handicapped, physically and mentally and so on.

Figure 9 describes the activities of a middle class adult who wants to live by himself. This strata may only be depicted through directed graphs applying weights to describe things like accomplishing an activity by oneself, with others, with help or giving assistance.

It would be of the utmost importance to implement these design processes for all the architectonic elements, namely, structural system, environmental comfort, building systems, etc.

I hope the more detailed analysis of the design processes of the architectonic element called activities will throw light on the dimension of this undertaking. First I will characterize partially the core construct of the MPSTW/SGPGM and later on I will introduce terminology that will enable us to perceive how all discussions reflect only in terms of collections of these fundamental "core

construct"in this homo-iconic system to enhance urban sustainability through a computer revolution.

The substance of the form (Figure 10) shows through its topological relationships how strongly the intra-apartment conditions are dependent on the inter-environment conditions.

Or how "form follows function", I mean how function is transformed into meaningful forms such as eating area, sleeping area, leisure area, circulating area, hygienization area, etc inside the apartment world. It shows how the latter are thoroughly dependent on corresponding social and environmental structures.

The detailed sustainable sanitation systems shows clearly the connection of the apartment to the surroundings where rainwater can be stored in lakes or in the roof or tanks, infiltrated through swales with biologically active soil in the ground, recharging the groundwater or respecting the local water cycle (stormwater runoff is often loaded with a wide variety of organic and inorganic chemicals, hence direct infiltration into the soil should be avoided); separated blackwater can be treated anaerobically in biogas plants combined with the digestion of organic household

wastes results in a mixture that is suitable for this process; usage of an existing treatment plant without nutrient removal as irrigation water that carries also fertilizer demands integration of agriculture and landscape into cities; hygieniezation of the effluent or crop restrictions may be necessary; treatment of wastewater or greywater from bathrooms, washing machines and kitchen (little nutrients) can be done by constructed wetland as wastewater lagoons or sandfilters with reed; the combination of treatment and agriculture can be applied with the system of energy forests; composting provides withlong-term fertilizer while biogas-systems or aerobic wastewater

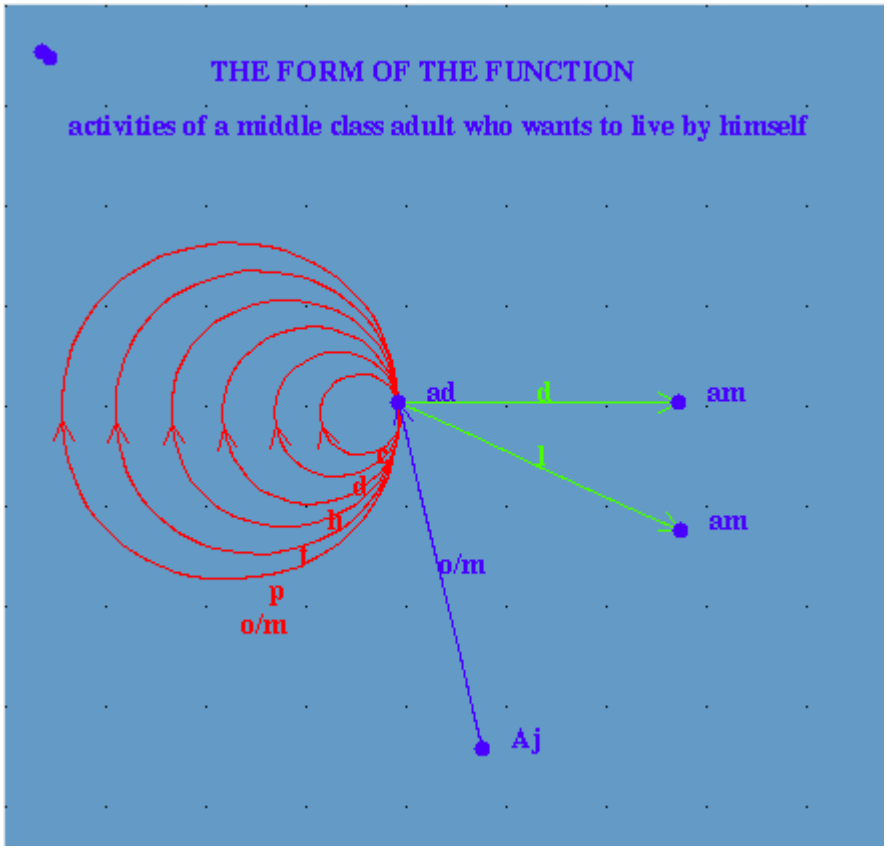


Figura 9.

treatment produces fertilizer that should be applied during the growth periods only; etc. [OAO98].

For such a tight integration of sanitation systems of the apartment with the environment, one is already modeling the Tertiary Waves of the ecodesign model strongly. Hence evaluation of overall efficiency with tools such as LCA(lifecycle assessment) MIPS (material intensity per service unit) or SPI (Sustainability Index) may be unfolded in the Department of Energy under Prof. Lineu Belico's coordination.

A more encompassing notion to retrofit the wastes reaching the goal of zero emissions is the Integrated Biosystem. For a biologist, an integrated biosystem contains at least two biological activities or subsystems where nutrients in by-products (waste) from one sub-system serve as resources or inputs for another.

The integrated bio-systems approach follows three basic principles. The first principle is to use all biological organic materials and wastes instead of throwing it away.l The second principle is to obtain at least two products from a waste. The third principle is to close the loop for the material and nutrient flows to achieve total use of a resource and zero waste disposal. Its application ranges from situations where natural resources were limited and when the full use of resources is crucially interlinked with human survival, problems related to waste management and to improve industrial productivity to utilization and management of agro-industrial wates in industry [Foo00]

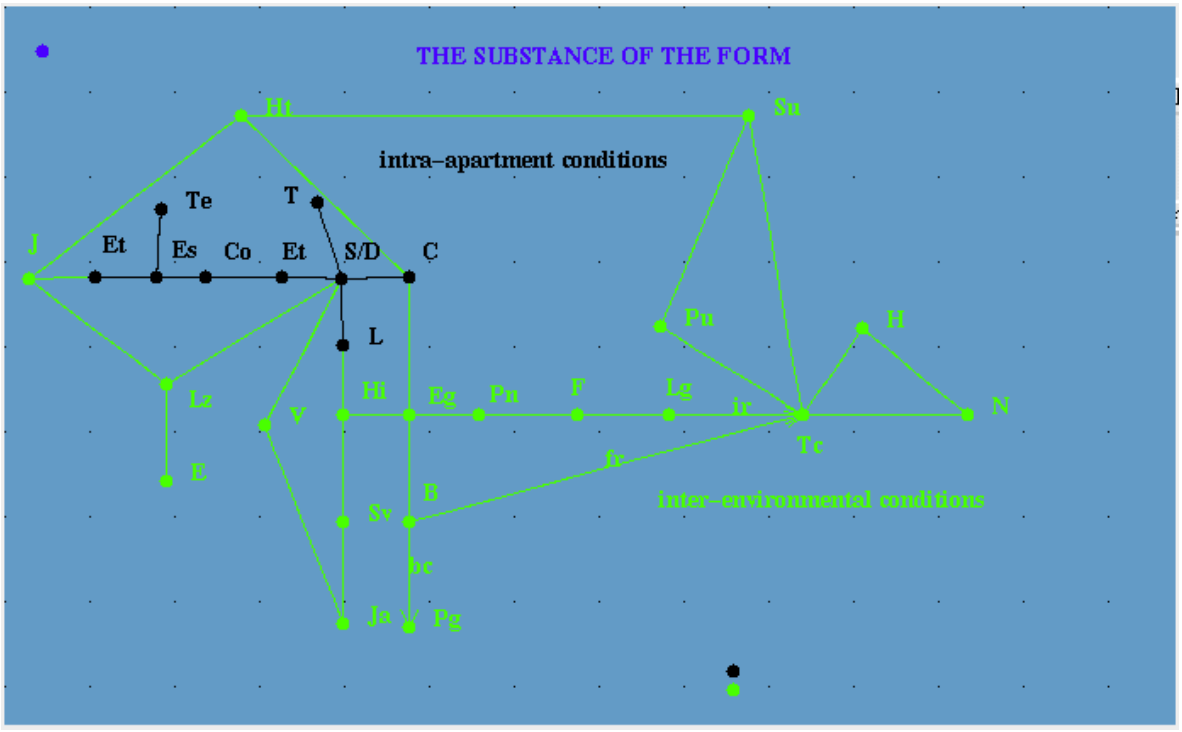


Figure 10.

B – biodigestor bc – biofuel C – eating area Co – corridor D – adult sleeping area E – sport area Eg – sewage Es – ladder Et – entrance area F – sand filter fr – fertilizer H – local agricultural area ir – irrigation J – garden Ja – greenery in wall L – lavabo Hi – Hygiiezation Lg – lake Lz – leisure area N – nature o/m – organization and maintaining area P – circulation area Pg – gasoline station Pn – natural or built wetland Pu – cattle S – social area Se- living room Su – supermarket Sv – laundry T- working area Tc – agricultural land Te – terrace V - varanda

The description above shows clearly the intertwining of microworld and macroworld. Moreover the MPSTW (Figure 1) describes the structure of the microworld. The model generates the macroworld from this core, hence shaping a homo-iconic

system: it consists of structures built from a single type of construct. All objects within the system have identical implicit semantics. Its sense of object is determined by a system of regional categories, namely, the processes of the interaction of the object with the environment: homeostasis, continuity, differentiation, repeatability and the design processes, namely the substance of the function, the form of the function, the substance of the form, the form of the form. Subsuming the phenomena of the considered region under these categories transform them into experience objects. Based on the principle that states the conformity to the things themselves or its semiotic value better explained in [Lour00:70] and briefly related to in the attached paper [Lour01b], the model does not refer directly to phenomena, but only indirectly (mediating it) through the categories that subsume them. This factoring likens to its factoring legitimation. It clearly shows that they deploy an internal diversity due to the meaning of the regional categories.

Thom, the author of the catastrophe theory insists that here one must deploy the meaning of the regional categories by explicit

constructions. I show how graph theory eases the transition to the geometrisation of concepts. Thom insists that here one must spatialize the concepts insofar as to employ resources from geometric description – only this allows for true objectivation (Figure 2a) and 2b)).

I must call attention here that the geometric modeling must be isomorphic to this conception. And indeed inspired by the graphic Dutch artist M.C. Escher's tilings and his breakthrough call the prototile I managed to translate the gist of the sustainable function into form.

This means I am managing to discretize an experience opening the gate to a scientific basis to plan sustainable cities.

The morphological turn highlights that the shape of the world as a form of language is not an invention of human subjectivity; to the contrary it is the objectivity of the real that unravels a formidable demand for ontological stability.

The structuration of the world in Gestalten and in states of perceptible things and linguistically describable is an objective structuration, even if of a qualitative nature.

The process of recognizing the shapes precedes the language level and conditions the possibility of revealing it [Coco92].

I will delve into this deeper in the section I.4 The Symmetry Group of the PlaneGeometric Model.

Hence MPSYW/ SGPGM is a homo-iconic system. A system that reflects upon a homo-iconic system therefore reasons about structures of this single construct. All reflective discussions exist only in terms of collections of these fundamental objects in a homo-iconic system.

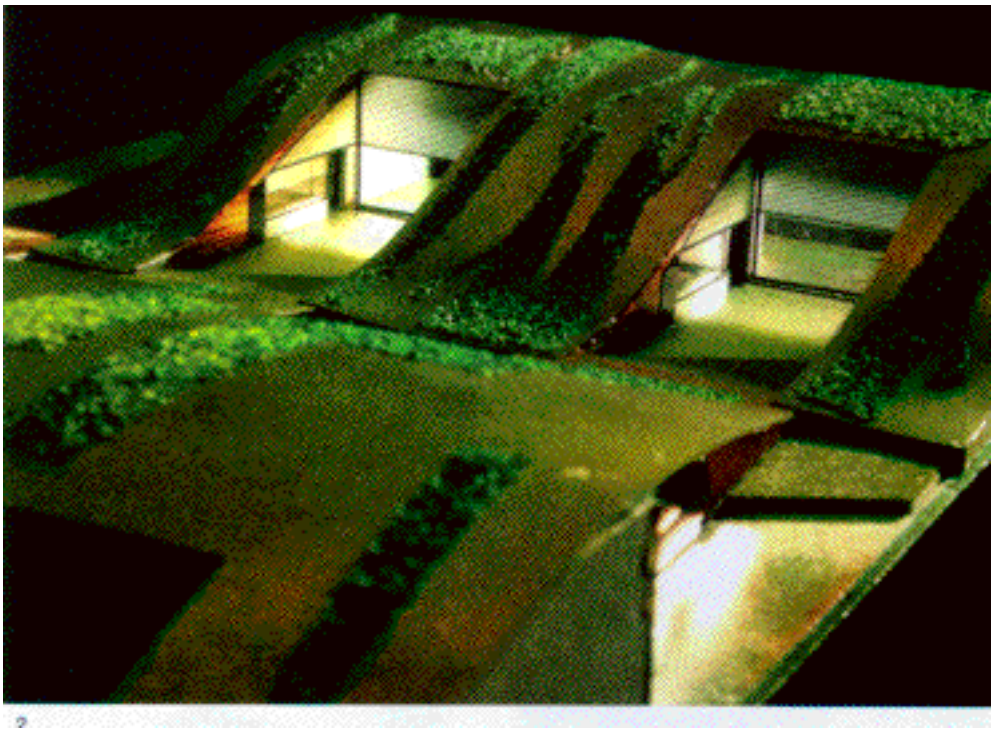
In terms of the MPSTW and its underlying geometric model SGPGM, thanks to the basic core of the MPSTW namely its processes and the respective isomorphic geometric processes, the urban ecosystem unfolds as a single organism, an autopoietic entity that is distributed in time and space by recursive partitioning into parts that are conceived similarly structurally speaking to tune in within the whole: Mother Earth, ecocontinents, bio-regions, ecocities, econeighborhoods, ecobuildings, etc. If built, these parts behave like autopoietic systems or machines mimicking the behaviour of the living being.

The design of the City Fruitful in Dordrecht, Netherlands (Figure 11) illustrates the idea of

an integrated biosystem. Its design aims to combine the principles of energy, ecology, economy and emotion. The site is bounded by an inhabited circular city wall and contains 1700 homes (eighty percent low rise), 24 hectares of land cultivated under glass and over 5 hectares of open cultivated land. This surprising combination couples two major consumers of power: glasshouse horticulture and housing. The houses are situated so that they can be heated by the warm, oxygen-rich air emerging from the glasshouses. Since the houses are built between the warm glasshouses, they will cool less quickly than normal. Windmills and solar panels can generate the rest of the power needed. As a water-saving device, the plan includes the Water Castle. This is primarily a reservoir for periods of drought, but the water can also be used to generate power or as a supply for the fountains. Hydroponic glasshouses are proposed for the treatment of waste water, since these can take advantage of solar heating to energize an intensive biological water purification process. After passing

through basins with plant roots, bacteria, algae and various plants, the water can be discharged into the reed-bed.

Finally, priority is given to double land use. Glasshouses are therefore stacked above or below the dwellings. Not only do they thus insulate the dwellings, but they provide employment opportunities very close to home. Moreover, the residents will no longer have to travel to the shopping center for fruit and vegetables and, despite the high building density, they will always live above or below a mass of greenery.



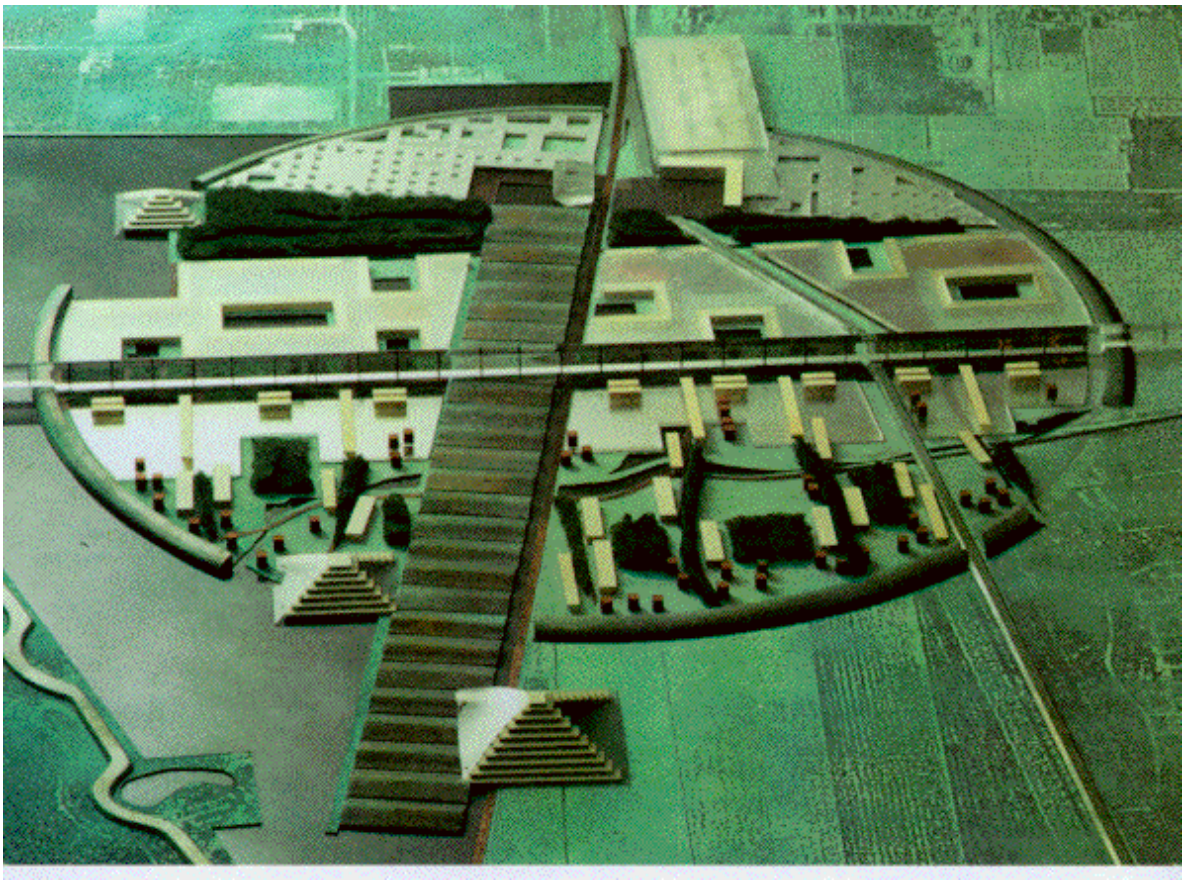


Figure ll. City Fruitful, Dordrecht (NL). Kuiper Compagnons Bureau voor Ruimtelijke Ordening en Architectuur B.V. and Oosterhuis associates, Rotterdam. Source: (Mele99]. All case studies concerned with ecocities stick to Euclidean Geometry. This is also the case here. I deem this as the most creative example from the form viewpoint for cities.

From the viewpoint of rational coherence of the ecodesign model, the new developments in sustainable architecture and urban design trigger the concern with the urban design and planning as well as with the three dimensional aspect of the project.

To subsume these features under the interaction processes of the architectonic object with the environment and the design processes demands a thorough knowledge of the integrated biosystems, energetic systems and sanitation systems. The contact with the Department of Energy from Polytechnic School may enhance this trend. Professor Dr. Pedro Sanches from the Department of Electronic Engineering Systems agrees in developing the Electrical systems based on sensors and actuators to control the temperature. Prof. Dr. Marcelo

Knörich Zuffo is willing to cooperate with the virtual reality representation of the ecobuilding three-dimensionally. Obviously this technique will greatly ease the vertical integration of the architectonic elements. At this phase one cannot forget about three-dimensionality.

19

IV.3 Widening the reflective nature of the MPSTW/SGPGM

Paving the way for the reader to have a glimpse of the complexity of the task I am expected to advance, I will borrow from the mathematician Brian Rice's The Arrow System [Rice01] some terminology related to ontologies. I hope the reader realizes the isomorphism of conceptualization within the context of the domain model and the conceptualization introduced by aspect oriented programming in chapter III. I would like to remind the reader my intention of building a multi-paradigm based knowledge system. What was introduced in aspect oriented programming before is restricted to the implementation code. I highlighted the importance to do the same in the domain level and software

[19] As I have already put forward, it seems the experts in computer science are not at all concerned with terminology. The same words are used referring to things with different granularity.

architecture level. The need to sharpen terminology will be attempted next year.

First Rice defines **context**. It is a function of a model that provides an environment for agents that fully supports that model. A context therefore provides all accessible information in terms of structures specified by that model.

An **ontology** is an explicit formal specification of a conceptualization, or a conceptual model for a domain. Specifically ontologies are concerned with the objects, concepts and relationships among them within some domain. In terms of contexts, an ontology is associated with the space in which actors model domains in terms of the ontology. Hence the ontology for a context is a description of the objects that are available for discussion.

An **ontological frame** denotes the formal model specified by a structure of ontologies, as well as the universe of discussion that it generates. Frames are structures of contexts, perhaps uncountable in size. The ontologies specified for the frame may crosscut each other in arbitrary ways, to allow the frame user to have the structure due to crosscutting available for study and first-order use. A model-level system develops its knowledge in terms of ontologies and their frames. Often, designers may find models within such a system mutually contradictory, and this is both permissible and desirable. For example the unpredictable need to assimilate the breakthroughs in **smart buildings** and **energy-generating buildings** based on exploiting the third dimension. Contradictions between models in the system allow for designers of the system to make assumptions contrary to the current state of information and study the consequences of the assumptions upon the relationships among various models.

In this case, the methodology based on the "prototile", a tile of different format [Lour98] and crystallographic groups of the plane would not yield a true "free plan" in an infinite sense ranging from a single room of a house, its internal conformance to other rooms once it is generated with a free shape due to being based on modeling the ecology of the human behaviour in interaction with furniture, the dialogue of the form of the house with the form of other houses or differentiated buildings and urban facilities to the coordinated evolutive shape of a city. This necessarily entails to take into account threedimensionality. The burden of threedimensionality in a single bioclimatic building could well be expressed with the cited methodology.

The need for integrating all sorts of human groupings respecting their behaviour expressed along the human life cycle calls for a more robust technology. This was solved with the introduction of the **subgroup relationships of the crystallographic groups** (Figure 7). It was not possible to apply the concept to architecture straightforwardly. The poster was developed with simple geometric motives to show how the morphodynamic level would behave. For our surprise it presented many challenges and a real world of form with specific laws emerged. Curiously it likens to musical scales. But also it unravels a spatiality where three-dimensionality seems tameable.

The only means to be beyond the challenges and apparent contradictions is merely to reify the instantiation of a new context that distinguishes the new set of inconsistent information from the old. Here the interdisciplinary clash between algebraic ways of dealing with crystallographic groups and the sensitive geometric way inspired by Escher and the geometers reaches a climax. This will be shown with more detail in section I.4 .

Indeed I have been weaving arguments such as categorical perception, object identity and position identity for the reader to understand that although an architectural language in

general terms can be likened to a natural language, indeed it is far more complex and its nature is rather likened to musical language. Hence it presents melody and harmony. The **architectural chords** like the musical chords can be played one after the other at the conceptual phase, a very talented designer would play them together but at the building phase the **architectural chords** are manifested "played" together or composed simultaneously in the shape of the architectonic building.

It is our hypothesis that **the subgroup relationships of the crystallographic groups** ease the introduction of **"harmony "** into architecture, but this does not happen without conditions that are examined in the next section. And these conditions are dependent on concepts such as categorical perception and object identity and position identity.

These concepts are also the bottleneck in class-based languages and prototype-based languages in the Object Oriented Paradigm in computer science. To describe common behaviour of objects, it is necessary to divide them into different categories determining this common behaviour. The category musical chords for example is a combination of three or more usually concordant tones sounded simultaneously.

A category may be defined by the concrete behaviour of its objects, or alternatively based on an abstract notion that separates it from other categories. Creating an object by concrete categorization therefore means to copy the concrete behaviour of one or several other objects.

Abstract categorization is the result of an achievement of abstraction by the inventor of the abstract category, thus distinguishing it from other abstract categories. An abstract category therefore is independent of the concrete behaviour of its objects at any time.

While a concrete category is determined by the modeled behaviour of its members, an abstract category determines part of the behaviour of its members. Abstract categorization

of behaviour may lead to different categories which determine different behaviour nevertheless based on a common behaviour pattern. Additionally, abstract categorization in contrast to concrete categorization is not related to the entire behaviour of its members. Hence through abstract categorization once cannot unravel the true nature of the thing. It cannot exhaust its semiotic value.[KK95].

Obviously the interface that emerges in the transition from a crystallographic group to one of its subgroups is of the utmost importance in architecture because it eases the concrete categorization process. Here the architectural chords become meaningful. However this interface is extremely volatile described by the organic approach from the algebraist Moser used to unfold the poster represented in Figure 7. Simply because Moser's approach does not enable you to calculate the measure of the sides of the fundamental region. So when a chord is shaped it cannot be repeated again! It is necessary precise musical notation to reproduce it again! This will be treated in section IV.4.

But here it is important to introduce the topological and geometric notion of position identity. This begs a question: May a position identity reveal an object identity? Under which conditions? Indeed similar conceptualization pervades computer science applied to many volatile concepts as shown in Chapter III..

Obviously our highly compartmentalized and fragmented world makes it difficult to understand this sort of reasoning, a far cry from the quantitative world.

Brian Rice gracefully warns us about the nature of what I have been trying to convey. Ontologies are not absolute and calls for the concept of ontological relativism.

It is an underpinning of human understanding to approach ontology constructions with some frame in mind. This frame may be likened to what Peter Naur calls **Theory Building View** [Lour01b] and is responsible for the need to work in close contact with those who

possess the theory of the idea notably in activities characterized by artefactual building such as art, craftsmanship, programming, mathematics following the nature of catastrophe theory and so on.

This hermeneutic cognitive process highlights the fact that no ontology can be inherently optimal for working with a specific domain. *No knowledge is an island*, insists Brian Rice. Yet exactly due to the inability of humans to devise whole horizons, the current practice of education is still one of compartmentalization making almost impossible inter-, multi- and transdisciplinarity [Lour01c].

Being this the state of things, I would call the reader's attention to the need of deploying paradigms that foresee the consequences of the constituency of human cognitive processes and take into account the need to have a general frame of mind that would adjust to the relativity of ontologies.

Considering the relativity of ontologies, then, the system should not consider a particular ontology or frame as an absolute reference, but instead a relative reference where the results achieved by the modeling by default apply locally unless proven otherwise.

Peirce's general theory of the sign [Lour01b], [Lour00] in essence express this truth. Obviously the way the **MPSTW/SGPGM** unfolds takes this into account. So inherently there is no absolute contradiction in the fact that its unfolding now has been opening new trails in the dense forest of knowledge.

Brian Rice's research with reflection has a Peircean flavour curiously. His model-reflective system may represent the user as an agent, making logical guesses about the user ontology frame by the interactions provided. If the user exhibits contradictory ontologies, then the system will update the frame accordingly.

In this way, an agent may study the interaction of these ontologies. This allows for two or more users to interact with the system via the same device with no secure identification made, while the interactions between their beliefs and desires are preserved and managed. It similarly provides for the use and analysis of a user history in tracking the beliefs and preferences of the user as system knowledge develops. The system provides all of these benefits through casting users, external software agents an all other incoming information in terms of agents and their ontologies.

Briefly he dissolves elegantly a state known as subject-object dualism. So humans are not radically different from or better than other beings [Lour01b].

As I put forward when emphasizing Peirce's general theory of the sign, if one does not open the black box representing the transcendental reasons tapped to his Dynamic object of the sign as a complement to its Immanent Object (scientific aspects), all human activities will be dependent on pupils having good masters to evolve and trigger synergies. And obviously the explosive growth of population does not cope with this low rhythm of education. So Rice's reflective system is a real must, if one believes in creating Paradise on Mother Earth. The reader may visit it on http://tunes.org/ search for Arrow.

Crosscutting of ontologies is the process of taking a given domain and interpreting it in two or more mutually independent ontologies. **MPSTW/SGPGM** is a crucible where this

process is intense. It approaches the ideal model-reflective system which provides the arbitray crosscutting of ontologies to maximize information accessibility while allowing arbitrary ontologies for use in the desired domain. This essentially provides the system with definitions for the elements of a domain from various perspectives, so that the reasoning structures with access to the domain knowledge can analyze the system from the currently optimal perspective.

The subgroup relationships of the crystallographic groups represent one of such powerfully expressive reasoning structures.

The next section emphasizes the role of the subgroup relationships of the crystallographic groups to ease the vertical integration of the architectonic elements .

IV.4 The symmetry groups of the plane geometric model (SGPGM)

I will reproduce here what Gregor Kiczales et al explained about aspect oriented programming and try to map isomorphically his reasoning structures to the domain model and its underlying geometric model.

A component, if it can be cleanly encapsulated in a generalized procedure (i.e., object, method, procedure, API). By cleanly, he means well localized and easily accessed and composed as necessary. Components tend to be units of the system's functional decomposition such as bank accounts, and so on.

An architectonic element such as activities, structures, environmental comfort, energy-generating structures, etc are the components of my ecodesign model.

An aspect, if it can not be cleanly encapsulated in a generalized procedure. Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways.

All the interactions among the processes play the role of aspects which represent real cross-cutting concerns. The architectonic chords were likened to musical chords and are expressed through the geometric model.

A GP-based implementation of an application consists of:

(iv) a language,

(v) a compiler (or interpreter) for that language,

(vi) a program written in the language that implements the application.

A domain model of an application consists of:

(i) a domain model here the MPSTW

(ii) the designer's mind [20]

(iii) the geometric model orbiting around the symmetry groups of the plane and the dotless plane in this case

Likewise an AOP-based implementation of an application consists of:

(i.a) a component language with which to program the components

(i.b) one or more aspect languages with which to program the aspects

(II) an aspect weaver for the combined languages

[20] I highlighted the importance of a prototype-based programming language like Self because it allows a symbiosis between the computer and the human mind. This paradigm of interaction together with the cooperative work through the multi-user programmable virtual reality called Kansas in Self mimic the cognitive processes involved in architectural design perfectly well.

(iii.a) a component program that implements the components using the component language, and

(iii.b) one or more aspect programs that implement the aspects using the aspect languages.

Likewise a symmetry-group-based implementation of the ecodesign model consists of:

(i.a) a multi-paradigm based ecodesign model with which to model the architectonic elements

(i.b) symmetry groups of the plane (rosette groups, frieze groups and crystallographic groups of the plane, similarity and conform groups of the dotless plane) with which to integrate or compose the architectonic elements

(II) a geometric weaver to compose the architectonic chords (role of harmony or vertical manifestations of music) through the subgroup relationships of the crystallographic groups and coordinate the dialogue of the free plan with the neighboring plans conforming to a smooth form metamorphosis

(iii.a) a Self program that implements the architectonic elements and

(iii.b) a multi-paradigm based program that implement the interactions, cooperative work and cross-cutting concerns [21]

The real bottleneck here is the intense inter-, multi- e transdisciplinary contact to deal with interactions and especially cross-cutting concerns. These are entities that have no

[21] I hope the Self/R being developed by Jecel de Assumpção Mattos fulfills this need. See http://www.merlintec.com:8080/software, it integrates reflective abilities and aspects.

object identity. One of the concerns in Self is to treat them as having object identity or to treat them as objects.

It is impossible to build an intelligent program to mimic the cognitive processes involved in the confection of the poster from figure 7, without uncovering an approach that crystallizes it or turn it into an object, a thing that can be handled. My idea to accomplish this was to fuse Moser's PhD approach with Schwarzenberger's concern.

I spent fifteen days in Brasília in April-May exchanging ideas with Prof. Norai Romeu Rocco, Director of the Mathematics Institute from Brasília University discussing this volatile "entity".

Now the interdisciplinary problem happens. He translated my concerns according to his attached report. Although I intend to implement my ecodesign model and its underlying geometric model, I strive to do this in such a manner that a human mind can also do it in case there is no computer. Even if this takes time and is exhausting.

Of course my research is not viable for all social classes without the use of the computer. However to mimic the human cognitive processes of the designer is a must if one wants the knowledge system to behave as a semiotic machine. Or mimic human mind.

Delving deeper into semiotics one realizes how fundamental for sustainable development is the semiosis between machines and minds.

Winfried Nöth argues [Nöth02] that it is difficult to show where semiosis in the sense of evolved humans is really reproduced by the human brain without the help of a simple pen for example or a printed book and where the machines are not really endowed of human reasoning.

What I am trying to do is exactly "very human software" for the sake of mankind's survival!

Although dialoguing with him I perceived that this approach is really evolutive due to the recursive nature of the crystallographic groups, I mean one can generate an infinite number of crystallographic groups because each subgroup may become a parent group and then one can calculate its subgroups. This is really what a sustainable designer needs to build evolutive cities.

The problem with GAP is that it gives the generators of these subgroups but not the relationships that help to define the geometric shape.

The problem in dealing with algebraists is that they do not care about the morphodynamic level and have no sensibility to understand the world of form.

In a dialogue with Jecel Mattos de Assumpção, who helps me with the implementation of my ecodesign model in Self it is evident that the results achieved in my PHD dissertation are powerful and efficient. However the expressiveness of the computer to accomplish the goal is still poor as one realizes when one examines GAP or MAPLE. Without powerful graphical user interfaces like those we can implement in Self one cannot go much further.

And yet it requires lots of investment to start a design team for building a sustainable city.[22]

Indeed I have been studying musical perception and musical theory for two years to be able to delve deeper into morphogenesis.

This study requires at least a very nice keyboard and LOTS OF TIME to deal with musical scales. I barely had time to attend the music classes at the Music Department from ECA-USP under the direction of a professor that is a talented conductor!

This step of the research demands mastery of artistic cognitive processes and is incompatible with the intense intellectual activity and exchange of information with

[22] W hen I see luxurious ships and planes, I realize sustainable cities are feasible. As Scandinavians put forward the problem is not money, it is of technological knowledge and of course the change to sustainable development as a first priority.

researchers abroad that I underwent this year. It can be likened to my activities in the first year of the Post-doctorate.

This may seem for a non-mystical researcher like loss of time. I hope I have persuaded the reader that while I spent my time delving deeper into hermeneutic computer science, the international community in computer science made great progress towards hermeneutic computer science even if they do not use this term. **Agile methodologies** [Cock00] especially **X-programming** [Beck00] liken software development to art and most of the software developers dealing with X-programming are also jazz musicians and liken the latter to jazz.

Since I am mystics, I am aware of my action in astral body. This stuff sounds like mysterious for those that at least respect there are different people on Mother Earth but for reactionary minds like those in Medieval Age this sounds like crazyness. Well, the post-quantum Mechanics from Jack Sarfatti guarantees the speed of thought transmission is above the light speed. So I see no mystery that now I have in my hands after having gone abroad twice to Budapest and Vienna the intellectual output from computer scientists in the last two years. Of course in a state like São Paulo where the financing agency called FAPESP does not recognize the importance of joining Workshops held by the most talented researchers and the attendance of conferences where these people appear as invited speakers and in general do not publish their talks is not considered a first-order activity. For the computer scientists from the First World countries Workshops are the centers of thoughts of the conferences and to have a Workshop accepted in conferences like ECOOP, OOPSLA, ESEC , etc the organizers must have an excellent curriculum and propose only challenging topics. Moreover the referees are known. They favour transparency. These

referees have home pages and e-mails and do their best to divulgate their research and respond immediately every message they receive.

Most of the time since I work inter, multi and transdisciplinarily my knowledge in mathematics and computer science is considered weak by totally anonymous referees from FAPESP.

However I dialogue with the most talented researchers in mathematics and computer science and to dialogue obviously one is at the same level, on the other hand, the dialogue is not possible and they get impressed how I managed to develop such a research in a country like Brazil!

Needless to say that nowhere in the world similar research is being held in terms of sustainable cities. All that is being done is obviously expensive "mock-ups" that still mimic the old times of promotional architecture.

The sensational implosion of the Trade World Center may serve as a lesson for those that challenge the laws of the environment, of social integration and the needs of children and teenagers proposing solutions that indicate a real decline of the Western civilization.

In 1998, I had difficulties to argue intuitively about the possible destination of the Ecopolis, a 1 km-high and 500m-wide tower being built in Tokyo by Norman Foster if it got fire (Figure 12)

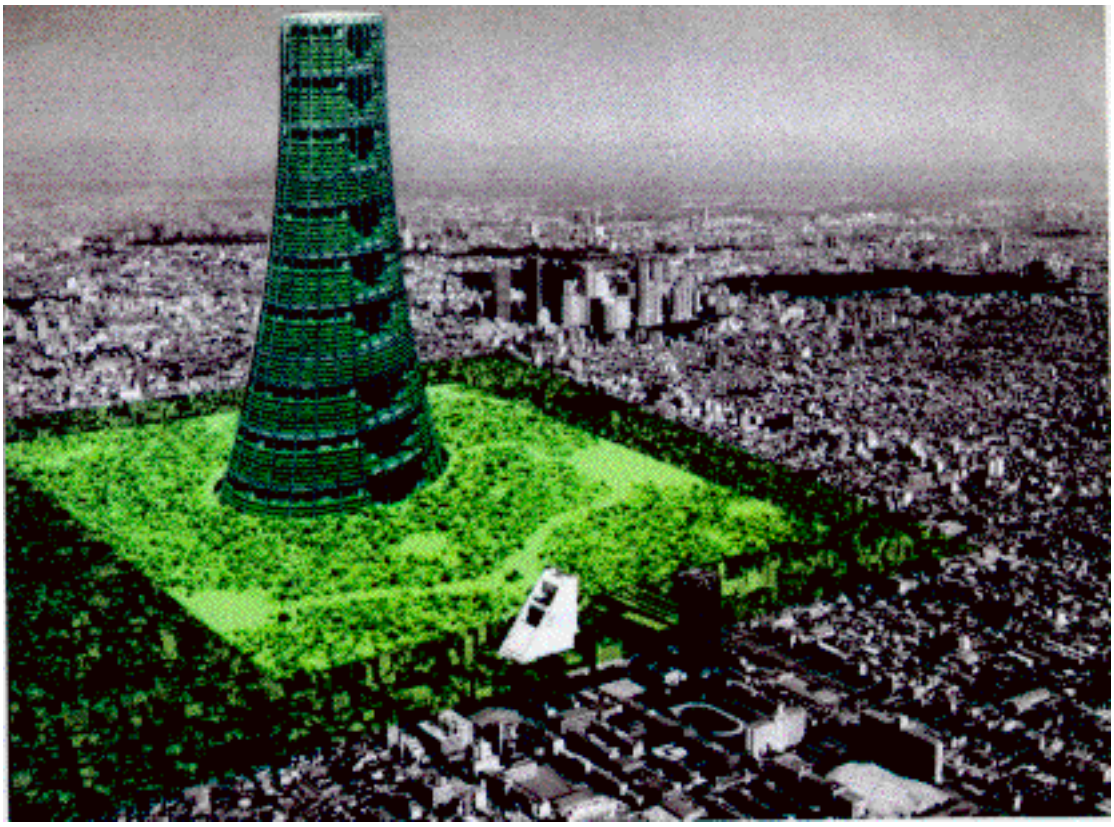


Figure 12. Are steel towers sustainable design?

Not only its building is an abuse in terms of energy and material use but also its destruction is obviously unsustainable in case of a disaster. This disaster must be taken into account in a life cycle analysis such those employed by the EMERGY method from the ecologist Howard Odum and the ecological economist Gonzague Pillet. Curiously Howard Odum's research do not receive grants in the USA.

Atual estágio da interação interdisciplinar entre o Prof. Norai Romeu Rocco e eu

Albertina,

Após a sua visita a Brasília, quando discutimos a aplicabilidade das idéias contidas no artigo de Schwarzenberger, parece ter ficado claro que esse procedimento exigiria um trabalho razoável de **representação matricial** dos grupos cristalográficos a fim de se explorar, no contexto do seu trabalho, um inter-relacionamento dinâmico (geométrico) entre um grupo *pai* e um seu subgrupo no processo de geração, por exemplo, do poster.

Quando digo **poster** quero me referir a todo o potencial elucidativo, emergencial e transformador que ele encerra, tanto no aspecto de *design,* de motivos arquitetônicos*,* etc., quanto no seu aspecto cognitivo.

A maneira como o Schwarzenberger introduz os elementos que o leva à classificação dos grupos cristalográficos planos – começando com um vetor de translação de comprimento mínimo e em seguida juntando a este uma sua imagem através de uma rotação de menor ângulo do grupo de ponto, de modo a produzir uma base conveniente do subgrupo das translações –, parece-me adequada para se atacar essa questão maior, da representação matricial dos grupos levando-se em conta o reticulado translacional correspondente. Entretanto, o trabalho de programação exigirá, além disso (como bem elucida o poster com os subgrupos do *pmg* e do *pm*), o aproveitamento desse

reticulado para, dinamicamente, gerar-se a região fundamental do subgrupo, H digamos, em função da correspondente região do grupo *pai* G, ao mesmo tempo em que esse tal subgrupo H seja representado pelas matrizes geradoras obtidas em função daquelas que geram G. Estou sendo o mais informal possível, para evitar a introdução de termos mais técnicos.

Mas é importante recapitular que os elementos de um tal grupo cristalográfico plano, G, são pares g = (v, φ), em que v representa uma translação e φ uma transformação ortogonal do plano (rotação ou reflexão, no caso). Assim, a ação de um tal elemento sobre um ponto (vetor) x qualquer do plano é dada por (v, φ)(x) = v + φ(x). Isto significa que a representação matricial de (v, φ) é dada por uma matriz *afim*, isto é, uma matriz que envolve a parte translacional v e a parte linear (ortogonal) φ.

É, portanto, uma matriz da forma $\begin{bmatrix} M & v \\ 0 & 1 \end{bmatrix}$, onde M indica uma matriz 2x2 que representa φ, numa base linear adequada do plano euclidiano, enquanto o 0 (zero) que aí aparece indica o vetor linha (0,0). Neste caso, o ponto x é representado por um vetor coluna da forma $\begin{bmatrix} x \\ 1 \end{bmatrix}$ para efeito da multiplicação das matrizes: $\begin{bmatrix} M & v \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} Mx + v \\ 1 \end{bmatrix}$. É claro que uma matriz da forma $\begin{bmatrix} M & v \\ 0 & 1 \end{bmatrix}$

será inversível sempre que a matriz M o for. Neste caso a inversa é dada por $\begin{bmatrix} M & v \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} M^{-1} & -M^{-1}v \\ 0 & 1 \end{bmatrix}$. Esta representação é apropriada pois ela encerra o essencial, sem necessidade de se explicitar o seu significado geométrico *a priori*, uma vez que a própria matriz diz isso implicitamente. Por exemplo, se estamos no caso mais simples, do grupo *p1,* em que apenas translações aparecem, os seus geradores são as matrizes $A=\begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix}$ e $B=\begin{bmatrix} 1 & 0 & c \\ 0 & 1 & d \\ 0 & 0 & 1 \end{bmatrix}$, correspondendo às translações pelos vetores $v=\begin{bmatrix} a \\ b \end{bmatrix}$ e $w=\begin{bmatrix} c \\ d \end{bmatrix}$ (linearmente independentes, claro). É imediato que AB=BA. Além disso temos $A^n = \begin{bmatrix} 1 & 0 & na \\ 0 & 1 & nb \\ 0 & 0 & 1 \end{bmatrix}$ para todo inteiro *n,* o mesmo se verificando para as potências inteiras de B. Isto quer dizer que A e B geram um subgrupo (do grupo das matrizes 3x3 invertíveis, com entradas reais) isomorfo ao grupo (aditivo) dos vetores da forma $m\begin{bmatrix} a \\ b \end{bmatrix}+n\begin{bmatrix} c \\ d \end{bmatrix}$, onde *m* e *n* são números inteiros arbitrários, este último sendo, precisamente, o grupo *p1* (com vetores básicos *v* e *w*). Ou seja, o subgrupo <A, B> é uma representação matricial fiel do grupo *p1*. Cabe observar que o *p1,* pela simplicidade de expressão de seus elementos como vetores de translação, dispensaria tal representação. Mas a situação é análoga

para qualquer outro grupo cristalográfico no plano. Por exemplo, se juntarmos uma *meia-volta* $T=\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ ao grupo *p1* obtemos uma representação do grupo *p2,* gerado então pelas matrizes A, B e T, cujas relações são facilmente verificadas: AB=BA, T^2=I (matriz identidade), $TAT=A^{-1}$ e $TBT=B^{-1}$.

Nas situações mais elementares dos grupos *p1* e *p2* acima, os vetores *v* e *w* podem ser escolhidos de qualquer maneira, desde que sejam linearmente independentes. Entretanto, se quisermos explorar os demais grupos cristalográficos precisamos impor condições mais favoráveis a essa base, levando em conta a métrica euclidiana do plano $\mathbf{R^2}$ (cf. Moser). Assim, por exemplo, considerando os vetores *v* e *w* perpendiculares, digamos $v = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ e $w = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ para simplificar, seja R a reflexão que fixa *w* e transforma *v* em *–v*. Então R tem a representação afim $R = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Se $X = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ e $Y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, então o subgrupo <X, Y, R>, gerado por X, Y e R, representa fielmente o grupo *pm.* Em lugar da reflexão R podemos também considerar a *glide-*

reflection $P = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$, obtendo-se assim as relações XY=YX, $P^2 = Y$ e $P^{-1} X P = X^{-1}$, de modo que o grupo <X, Y, P> é uma representação fiel do grupo *pg*. E, assim por diante.

Pois bem, voltando aos objetivos, vale a pena considerar alguns recursos, pois isso talvez interesse para a fase posterior, de programação. Como você bem sabe, aquele *site* http://www-sphys.unil.ch/escher/ proporciona um programa em *java-script* para gerar figuras usando os grupos cristalográficos. Parece-me que isto seria equivalente a uma primeira parte do que se pretende, que é gerar o reticulado (ou, equivalentemente, uma região fundamental) a partir da representação matricial (dos geradores) do grupo, com uma conveniente escolha da base (conforme descrita acima). Mas, como observei anteriormente, é preciso ir bem mais longe e talvez não se chegue de uma só vez ao objetivo. Nem poderíamos trabalhar com tamanha ambição nesta fase, principalmente quando se trata de um trabalho interdisciplinar como este. Você entende, além de eu não ser cristalógrafo, também não domino recursos de programação para tais propósitos. É um trabalho de pesquisa conjunta que exige ataques em frentes variadas, o que, por envolver diferentes especificidades, acaba consumindo mais tempo para se chegar aos resultados

desejados. Nesse sentido, em vez de tentarmos atacar a questão envolvendo um subgrupo arbitrário de um grupo *pai* G, talvez seja interessante em primeira aproximação considerarmos os subgrupos maximais do mesmo, isto é, subgrupos próprios H de G sem outro subgrupo intermediário. Digo isto porque em teoria dos grupos, como em qualquer outra área de estudos, você sabe, este procedimento está implícito no método reducionista. Isto certamente não irá interferir na visão holística da coisa que o projeto encerra como um todo. É apenas uma tática de ataque. Bem, eu penso em considerar os subgrupos maximais porque, por um lado, este é o primeiro passo, alternativo eu diria, rumo à questão, por exemplo, "**de como calcular o subgrupo *cm* como subgrupo do *cm* que é subgrupo do *pmg*** ?". Certamente esta questão deve ser entendida contextualizada, pois teoricamente isto é mais ou menos "elementar". De fato, a partir de uma *apresentação* do grupo *pai* G, que no nosso caso é um grupo finitamente gerado, dado qualquer subgrupo de índice finito H podemos encontrar uma apresentação de H. Este procedimento é algorítmico, conhecido como algorítmo de Reidemeister-Schreier para apresentação de subgrupos de índice finito de um grupo finitamente apresentado. Isto pode ser executado em computador, usando-se por exemplo o sistema **GAP** – Groups, Algorithms and Programming (http://www-gap.dcs.st-and.ac.uk/gap) –, que é um sistema para se programar e computar com estruturas

algébricas discretas, especializado em estrutura de grupo. Entretanto, o que nos interessa está um tanto além desse aspecto teórico, pois queremos explorar o aspecto geométrico, com ênfase maior no *design.* Neste ponto vale a pena levar em conta alguns recursos específicos do GAP, particularmente com relação aos grupos cristalográficos. Sim porque existe um pacote (*share package*) disponível no GAP, chamado **CrystGap**, desenvolvido por Bettina Eick, Franz Gähler e Werner Nickel, especializado em grupos cristalográficos em dimensões 2, 3 e 4, principalmente, cujos principais algorítmos são baseados no artigo [EGN97]. Nesse pacote os grupos já estão representados por matrizes afins, o que simplifica o trabalho. Além disso, um dos principais objetivos alí é descrever os subgrupos maximais de um dado grupo cristalográfico. Claro que o trabalho mais pesado está nas dimensões maiores, mas é de muita utilidade para o nosso caso, de dimensão 2. Em vista disso, acho que podemos nos valer desse instrumental para tentarmos avançar no que se pretende. Ou seja, tentarmos atacar primeiro a questão dos subgrupos maximais e analisar com eles a questão da transição do reticulado planar do grupo *pai* com aquele de um seu subgrupo (maximal). Se tivermos sucesso aí acho que teremos aprendido o caminho.

Como se diz lá no interior, “firma o pé, pois que ainda faltam uns dois tantos”.

Ou seja, isto que estou sugerindo pode ser uma linha de ação. Me parece segura, mas exige bastante perseverança. A começar pela parte de programação. Digo isto porque o GAP (sob o qual o CrystGap está implementado), embora disponha de muitas funções para teoria dos grupos, é um sistema desenhado para se computar em álgebra discreta; não tem recursos gráficos como os que o seu projeto demanda, pois não dispõe de aritmética de ponto flutuante. Na verdade isto não seria um grande problema, pois é possível desenvolver todo esse trabalho de forma descritiva, considerando-se as matrizes geradoras e os pontos reticulares com coordenadas racionais e estabelecer-se um subconjunto finito F,grande o suficiente, de elementos do grupo G, para se descrever uma parte significativa do reticulado planar. Para isso pode-se calcular as imagens de uma base adequada pelos elementos de F. Se H é um subgrupo de G, então em função dos geradores de H podemos destacar um conjunto equivalente F', e seguir o mesmo procedimento. Mas isso certamente não teria o efeito gráfico desejado. Daí a necessidade de se utilizar de outros recursos. Talvez nesse ponto uma saída, para uma primeira aproximação, seria aproveitar-se da descrição desses grupos em forma matricial e a partir daí, transferir os dados para algum sistema onde se tem recursos gráficos. Por exemplo, acho que o MAPLE poderia der uma opção. Digo isto porque, embora o Maple não dispõe dos recursos que tem o GAP

para computação com grupos, ele dispõe de muitos recursos gráficos e de um excelente pacote para álgebra linear e manipulação com matrizes, além de algumas funções para grupos de permutações e de uma linguagem de programação (Pascal-like, interpretada). Assim, com sorte pode-se "casar" as informações e chegar-se a algo interessante. Pode ser que numa primeira versão a coisa fique ainda um tanto braçal (bem menos do que o trabalho puramente artístico que voce experimentou para desenvolver aquele poster), mas eu creio que uma vez dominada a técnica pode-se avançar muito rumo à automatização do processo.

Bem, esta é a minha impressão do atual estágio desse projeto. Aguardo também a sua.

Noraí, em 07/10/2001.

Referências

1.Eick, B., Gaehler, F., and Nickel, W., *Computing Maximal Subgroups and Wyckoff Positions of Space Groups*, preprint.

2. GAP – Group Algorithms and Programming , Version 3.4.4 (1997).

M. Shoenert at al., Lehrstuhl D fuer Mathematik, RWTH Aachen. GAP can be obtained by anonymous ftp from ftp.math.rwth-aachen.de. See also on the World Wide Web on http://www-gap.dcs.st.-and.ac.uk/~gap/.

3. Moser, PhD thesis….

4. Schwartzenberger, ….

PART III

V. AN EVOLUTIVE MULTI-PARADIGM BASED KNOWLEDGE SYSTEM REASONS AS A SEMIOTIC, HERMENEUTIC AND AUTOPOIETIC MACHINE

RESEARCH PLAN
11/2001-10/2002
FAPESP PROCESS: 99/00333-0
Supervisor: Prof. João Antonio Zuffo
Post-doctorate researcher: Dr. Albertina Lourenci
Institution: Laboratory of Integrated Systems – Electronic Systems Engineering
Department- Polytehcnic School USP

Abstract. Times seems ripe to proceed to an implementation of an ecodesign model and its underlying geometric modeling that were conceived in a multi-paradigm design, namely application of catastrophe theory, Hjelmslev's semiotics, and graph theory and symmetry groups of the plane and the dotless plane to generate sustainable cities through computer. Agile methodologies, aspect oriented programming and the general trend towards E-types ease the task of unfolding multi-paradigm based knowledge system that enhances the human creative cognitive processes. Semiotics while a transdisciplinary vehicle is the glue that enables unity in the diversity.

INTRODUCTION AND JUSTIFICATION

The surprising and evolutive real world mimics the nature of the rainbow where each colour is a variant of the unifying paradigm of white colour.[23] Western people see the world through fragmented and compartmentalized vision due to the mainstream scientific emphasis applied to make inferences.

[23] Likewise here the idea of a multi-design paradigm lies inside the language like in C++ [Copl00].

Yet a semiotic understanding of the world like that unraveled through Peirce's eyes sees an esthetically good object as a successive multitude of inner and outer parts so related to one another as to impart a positive simple immediate quality to the totality. Paradoxically the post-quantum physicists also share the same vision. Thus the search for truth enables honest human beings to be beyond any ephemeral wrong vision (even if this lasts for centuries).

Jim Coplien's Multi-paradigm design [Copl00] is an amazing glimpse widening horizons in a mainstream computer science world stuck to details disconnected of the essence of life. Yet one cannot despise the power of reductionism because it is astonishingly a complementary process to holism!

Curiously the breakthrough introduced by aspect-oriented programming concerned with the multidimensional separation of concerns and based on reflective capabilities may open the gate to software that reasons as a semiotic, hermeneutic and autopoietic "machine" . Basically the ability to see a thing in its myriad of inner and outer parts unraveled through the infinite community of researchers enjoying themselves in a game whose only goal is to perceive infinity in the finite. Wow! This is the essence of Eastern reasoning!

Yes, this is the promise of the emergent **agile methodologies** if taken seriously. Wanting it or not, all computer scientists are trying to evolve in this direction through different paths as Lehman and Ramil put forward realistically [LR01].

Little by little computer scientists will join the pieces of a puzzle that is still totally disconnected.

I hope my research will contribute in this direction. More and more software developers are delving deeper into art, biological sciences and philosophy. Richard Gabriel [Gabr01] realizes programming is for all and must reflect friendly human capabilities.

Everybody speaks insofar this talent is genetically inherited even. Thus it is of the utmost importance to invest in a computational language to dialogue friendly with the computer and face the urgent problems that challenge mankind's survival.

Greimas's semiotics inspires me to deal with the theory of signs[24] rather as a theory of meaning that becomes operational when its analysis is accomplished at levels above and below the sign [GC79].

Greimas's semiotics believes that semiotics is not a theory of the signs, rather it is a theory of meaning that becomes operational when its analysis is elaborated at levels above and below the sign [GC79]. Obviously as Coplien advises this reasoning must be mapped from the real world to the domain model, software architecture and programming language.

Curiously there is no unique paradigm in semiotics that will realize this goal to model the phenomenal diversity. Up to now Hjelmslev's semiotics that enabled me to model the immanent aspects of the architectonic sign so well analysed in chapter IV intrinsically builds a bridge to the transcendent aspects of the sign. The latter is the realm of Peirce's general theory of the sign [Cola87], [Sant00].

However architecture is essentially the realm of shape and its morphodynamic level cannot be revealed without Petitot-Cocorda's semiotics that realizes the importance of catastrophe theory [Lour88], [Lour94] to deal with topological and relational entities [Coco92].

Likewise graph theory is unique in its properties to deal with abstraction and shape through planar graphs. Unfortunately natural languages are unable to reveal such realities as well as what goes on in music.

[24] The reader not initiated in semiotics may read sign as "synonym" for word. However he should ponder over what a word means. This is semiotics!

To study semiotics is above all to deal with powerful cognitive processes that are present in intelligent systems/machines [Nöth02] all over nature. [25] It will enable me to master their nature and hence introject them into software.

Obviously if the last two years are witnessing the need for change in the way of making software towards E-types, enhanced previous development in the design pattern community, assisted the fusion of different approaches such as adaptive programming, subjective programming, multidimensional separation of concerns, etc under the more encompassing aspect oriented programming, brought the emergence of agile methodologies, necessarily the feelings of the importance of delegation and the multi-paradigm design that should be applied to all levels of a knowledge system will be the next successful stage.

The chances of implementing my ecodesign model and its underlying geometric model to generate sustainable cities are becoming greater and greater in comparison to a computer science world dominated by S-types in 1988.

OBJECTIVES

The real bottleneck in software development up to now was the inability to deal with the multitude of inner and outer parts of an object as well as with the synergies generated by the endless game of objects and minds in the search for the infinite laws that rule these individual entities.

The worst consequence of this state of the art led to poor domain modeling. Fortunately real world demands curb the latter. The failures in developing and maintaining software to cope

[25] Like those uncovered in immunological system by the "consciousness scientist" Douglas Hofstadter [Lour00].

with an evolutive reality forced the community in a bottom-up way to react to narrow-minded specification.

The holistic ones are realizing the importance of the real world and its closest domain model whose main reasoning structures must be introjected into powerful architectures mirroring the phenomenological diversity. Unfortunately the true material in software is the code. It can be likened to the importance of the written word. However the writing ability is the last process in a successive sequence of steps that starts with the first cry when one is born and cannot unfold if emotional life is not taken into account. Hence the attempt to ignore the real source of inspiration in literature put forward by the poet-thing and metaphysicist Rilke [Maso74] as the infancy memories led in computer science metaphorically speaking to S-types[26].

Honestly if the whole community of researchers do not unite efforts to bridge this gap between domain experts and software developers, mankind will lag behind more and more smashed by unsurmountable problems that can only be tackled due to the mass scale with the aid of truly "semiotic machines". From my part I will continue investing in this, praying for every researcher to do the same insofar as the center of gravity changes from monodisciplinary excellence criteria to transdisciplinary criteria. When this point is reached, I am sure Paradise on Mother Earth will be emerging. And new institutions will rule the world tuning with true sustainability.

The real object of my research will be for the first time to face the challenge imposed by the new sustainable smart buildings and energy-generating buildings to deal with the three-dimensionality in the project.

[26] Specification types.

Since my approach is powerful, the master pillar to integrate three-dimensionality concerns with the bidimensionality of the free plan unfolded until now sucessfully lies in giving "object identity" to the transitional shape that emerges when one passes the border from one crystallographic group to another crystallographic group ruled by the subgroup relationships of the plane. To treat a "position identity" as an "object identity".

This adventure is impossible without multidisciplinary efforts. It has demanded lots of investigation in semiotics and computer science to realize what this meant. And my hypothesis to pursue this is to be able to measure the sides of the fundamental region. Unfortunately the algorithm to be discovered to realize this belongs to the interdisciplinary frontier of algebra and art. It is easier for me to learn the necessary algebraic instruments to apply to this goal than to expect a mathematician to do so. Here the mathematician is in the same place as the code implementers are. They are unable to see the higher level dimensions responsible for shaping meaning. [27] The worst is to spend a huge amount of time to convey this idea to people thoroughly blind to the morphodynamic level and its untameable difficulties.

In parallel one cannot ignore how to mirror this in continuing to advance domain modeling, software architecture modeling and code implementation modeling.

This means to apply Greimas's, Peirce's and Cocorda's semiotics to discover domain (ecodesign model + geometric model) and architectural patterns as mirroring reasoning structures of each other. Music theory guides the cross-cutting organization of these

[27] I am insisting on this point because it is dependent on the successful interaction between diversified cognitive processes that will enable us to find the right pieces of the puzzle that fit together. The pieces are already here but not enchained in a puzzle. The ability to do this does not require formal background but the ability to select in an ocean of information the right piece of information that resonates with the solution domain for this problem. This one does not learn at schools. It is life that teaches you! To the contrary the burdening of memory with unnecessary information disconnected of real life situation hinders you to develop these highly skilled abilities. When you manage to do it, the narrow-minded say: it is so easy! Yes, in theory it is easy to design a house! Try it! Especially a sustainable social building!

patterns. Musical chords and musical scales will be a hint to trigger insights to apply this to the emerging field of modeling a morphodynamic level. [28]

Another parallel thread is to continue delving deeper into recent research in the design pattern community. Until now no meaningful pattern to apply in my research was found simply because all their reasoning is hierarchical and does not tune at all with the bottom-up nature of architectural and urban design and the chosen programming language Self/Us [SU96].

David Lorenz [Lore97] shows how patterns can be used to describe the implementation of other patterns. He demonstrates how certain design patterns can describe their own design in a mirror manner. The process of assembling patterns by other patterns he names pattern tiling. He uses the Interpreter design pattern [GHJV95]] to instantiate several pattern tilings. Moreover he demonstrates how a tile can be flipped over, presenting tilings of non-physical objects such as design patterns as declarative entities that fit together without symmetry breaking or overlaps in an open-ended manner.

Basically a design pattern tells you how given a right problem to arrange your prototypes (objects and traits) in order to solve it. Conversely, an arrangement of prototypes can remind you of a pattern or induce you to create a new pattern, putting them in a totally new light.

However Lorenz's reasoning is based on hierarchy. And to understand the Interpreter pattern it is necessary to master the customary method of specifying the syntax of programming languages known as the Backus-Naur Form or BNF [Meye91]. [29]

---

[28] However to study musical theory without practice will not open the gate to form in terms of creativity. This takes time and demands patience above all.

[29] I have already a previous training in this when I was under the advising through e-mail by Joergen Lindskov Knudsen from the Computer Science Department – Aarhus University. Metaprogramming in the

Frank Buschman in his tutorial Patterns at Work claim that design patterns are language independent, however I still cannot see how in the above example.

Design patterns offer great interest because they work above all as conectors in software architectures.

The functionality of the domain model briefly described in chapter IV, although complex is expected to be mastered by any green architectural designer. Its interactive design environment allowing cooperative work partitions the task among many specific-domain experts such as structural engineers, environmental comfort experts, building systems' builders, etc according to the complexity of the sustainable architectonic object in its interactions with the environment to be designed.

Through it the urban ecosystem is modeled as a single organism, an autopoietic entity that is distributed in time and space by recursive partitioning into parts that are conceived similarly structurally speaking to tune in within the whole.

The parts into which the urban ecosystem is recursively partitioned include the concept of a sustainable planet, continent, country, bioregion, cities, boroughs, neighborhoods, ecobuildings, ecohouses, etc. Obviously a domain architecture for programming-in-the-large is a must to manage the resulting configurations described above. I expected Lorenz's tiling design patterns to fit into this reasoning. However the hierarchical nature of his reasoning has little to do with the arbitrary nature of each building in urban design, that may be a shop, an industry, a shopping mall, an artificial wetland integrated to the ecohydraulic installations and so on.

---

context of the Mjolner Beta Environment however is totally different from the exploratory environment from the prototype based programming language Self.

Above all they entail both urban design and planning. My architectonic object is open-ended . It shapes the urban ecosystem and is shaped by it!

This also demands simulations that do not rely on human interactive basis anymore, such as a simulation to determine sustainable materials to compose a layered-electric wall model, where the resistance and the capacitance of the materials are taken into account to enable the building to reach comfort temperatures with precision of air conditioning in the interior.

Such concerns call forth the introduction of parallel and concurrent algorithms as well as distributive systems.

Hence a domain and "software" architecture definition language (ADL) to model and structure these higher-level design concepts addressing the issue of evolution that requires high reflective abilities and composition and separation of different architectural concerns in conformance with the specializations of knowledge domains represents the next challenge to guarantee a full-fledged stable development of the knowledge system.

Faithfulness to the semiotic, hermeneutic and autopoietic reasoning must be enhanced in this level especially due to its overwhelming large scale needs. Here more encompassingly expressive support languages must serve as the explanatory tools of the **MPSTW** as well as common concerns such as features like printing, persistence, display capabilities; aspects like concurrency control, parallel algorithms and distribution, multiple views, configurations, layering, etc.

Recent breakthroughs represented by aspect-oriented programming and agile methodologies may ease the task.

Concerning the high-level mechanisms - those that specify the simultaneous interaction of several objects, classes, a system or a framework -while languages do not typically provide

the means for the user to develop higher-level abstractions, design patterns do [GHJV95], [BMRS+].

Experts interested in higher level lingual abstraction mechanisms[30]- have done little research to understand and promote the key concepts in component-oriented programming; that is, identifying what exactly is component-oriented programming and what language mechanisms exist for component-oriented style of programming and how to express these key ingredients in a component-oriented programming language. Those that try to express design patterns as language constructs build systems on top of existing programming languages especially C++. Needless to say this does not fit my needs at all.

Higher and lower level mechanisms (in relationship to the object, class functional unit in OO paradigm) are costly. And curiously powerful reflective abilities are necessarily to unfold these mechanisms. Reciprocally they also enhance reflective capabilities. Hence the introduction of aspect oriented programming may be a light at the end of the tunnel.

Obviously the sequence of the stages in the future research will happen according to an opportunistic reasoning. The trend that will prevail will be dependent on the conditions offered in the exchange with researchers from the Departments here at Polytechnic School and abroad.

MATERIAL AND METHODS

Although in essence my way of developing software tunes with X-programming, it necessarily entails a difference because my proposal of an evolutive multi-paradigm based

[30] many models for component-based software development, are based on sets of standards and frameworks (APIs)), and are implemented on top of a mainstream object-oriented programming language

knowledge system that reasons as a semiotic, hermeneutic and autopoietic "machine" is a complex undertaking.

Hence everything orbits around domain modeling. However this was conceived within the goal of being implemented by computer mimicking human cognitive processes involved in the design activity. All I knew is that procedural and logic languages would not implement it. But I read anxiously all developments in artificial intelligence that resonated with my ideas. And I was aware all the time that until 1987, computer science was not developed enough to start implementing my ideas.

But in 1988, the object-oriented paradigm was emerging as a powerful future direction in programming and as soon as I read about it, I realized it could match my needs.

Then I tested each language that was closer to my ecodesign model from C++ to Beta, till I met Self version 4.0 that obviously enabled me to keep faithful to the highly plastic, interactive and cooperative nature of my ecodesign model.

In the meantime while I kept discarding languages[31] and learning each day more and more about computer science, I unfolded the geometric modeling and the urban sustainability ideas. Obviously I knew how to cope with uncertainty in all fields wisely. Opportunistically I worked to the limits of my knowledge and searched around to see the changes. Through the development of my metaphysical cognitive processes I am aware that nothing resists an

[31] Curiously I avoided firmly to have a fellowship from FAPESP because I knew the undertaking was highly risky and the need to present scientific reports each six months would only make it difficult to justify in detail why a certain language did not fit my needs. Like agile methodologies I was aware it did not fit, justify it briefly to CNPq and changed direction as a true scientist works. Moreover CNPq offered the possibility for a sandwich PHD. Intense contact with the advsiser through e-mail made them realize how expensive and complex my research was in terms of computer equipment (CorelDraw, AutoCad, Photoshop and SunStation) and human resources (especially algebraists - preferably physicists because mathematicians do not make inference as a physicist does and reasons very differently from an artists- concerned with crystallographic groups hence with a nice geometric reasoning! And the necessary multidisciplinary needs to develop ideas about urban sustainability or access to Biological Sciences Department and Architectural Schools).

effort of concentration and that God does not invent a fish without the ocean! I mean one does not act alone in life.

Likewise, I would like to develop both the necessary knowledge to support robustly the unfolding of the knowledge system as well as keep an eye in an attempt to implement it in the style of X-programming. I mean to give a proof that your ideas are powerful. However the cornerstone here is the test.

Of course the first test is to be sure your ideas tunes with semiotic, hermeneutic and autopoietic machines, identifying the key reasoning structures that enable you to be sure you are faithful to the whole. Once you design this way, all the rest will be the same. For example in my PHD thesis I discovered the way of generating a free plan in terms of the apartment or any other building and how to apply crystallographic groups of the plane to generate the basic pavement.

Obviously the same methodology applies to all crystallographic groups once you exemplify through one.

Likewise the implementation in Self mimicked the cognitive processes of the artist based in freedom of creativity, interactive and cooperative skills.

The challenge in the postdoctorate studies is to integrate vertically and horizontally more complex buildings.

Hence a methodology based on the subgroup relationships of the crystallographic groups has become the next approach that extends further the first principles evolutively and once discovered the way to apply it to a defined architectonic object and its surrounding , all the urban design unfolds smoothly.

The crucible is to add measurement to the sides of the fundamental region through friendly algorithms that one can manipulate easily.

Once this algorithm is ready, I hope I am able to unfold simultaneously the algorithms concerned with separation and composition of concerns ranging from the domain model, passing through the software architecture to the implementation code.

Here the role of the test, adopting implementation code that mirrors the separation and composition of concerns already unfolded at the domain model (through processes of the interaction of the architectonic object with the environment and design processes and the reflective capabilities that allows you to reproduce the whole autopoietically) and its underlying geometric model that really enable the designer to build " cross-cutting architectonic chords" integrating all concerns vertically and horizontally and allowing this to happen in a cooperative game of invention, discovery where any designer can enter the game and any client sees his/her needs satisfied ensures this happens. If these requirements are validated, of course the development of a full-fledged knowledge system can proceed in a research team sheltering experts from very different fields to generate sustainable cities.

I believe there will be no difficulty in developing the diagrams concerned with separation and composition (especially the cross-cutting aspects) of concerns at the domain (ecodesign model and geometric model) and software architecture models.

However to continue further up to the implementation depends on the development of the Self/R programming language [Assu01].

Or of course the appearance in the scene of a multi-paradigm programming language that integrates delegation, the interaction and cooperation paradigm, reflective abilities and separation and composition of concerns sheltering the goals of aspect-oriented programming.

Material

Since FAPESP provides material through a project developed by the supervisor, due to the multidisciplinary nature of my research one cannot anticipate how this will happen.[32] Likewise the necessary need to go abroad and see and divulgate your emergent ideas is disregarded by FAPESP as an important investment. Curiously first world computer experts strongly highlight these exchanges in Workshops especially to try to introduce really E-types systems in computer science.

Consequently the way of proceeding with research this way requires the greatest courage and faith in God, that is enhanced due to my love to mankind and the pleasure of creating a better sustainable world for those that will inhabit Mother Earth in the near future.[33]

Research plan and schedule												
Lenght of a year (months)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Semiotics studies												
Algebraic algorithm development to add measures to the sides of the fundamental region of the crystallographic groups												
Cross-cutting concerns to integrate three-dimensional concerns triggered by energy-generating buildings and the												

[32] Until now I have been dependent on the good will of other luckier researchers that manage to tap technical resources and are generous enough to share them with the others. If FAPESP does not change its policy, talented people will continuously abandon public research. The most striking example is Jecel de Assumpção Jr who courageously and sustainably unfolds his research with the programming language Self away from the universities. Likewise the whole Self community works in entreprises abroad.

[33] Of course I feel reincarnation exists and hence I know I am preparing a better world for myself to live in it! This is also a powerful vision.

bidimensionality of the free plan												
Aspect-oriented programming concerns –traceability from domain model to code and vice-versa												
Development of ADL – architectural description language												
Implementation of a mini-project to test evolutive, semiotic, hermeneutic and autopoietic aspects												
Joining conference	**All year around**											

References

[AK00] Atkinson, C. and Kühne, T.: Separation of concerns through stratified architectures. Workshop on aspects and dimensions of concern: requirements on, and challenge problems for, advanced separation of concerns (orgs: Tarr, P, D'Hondt, M., Bergmans, L. and Lopes, C.V.) ECOOP 2000 Cannes France

[Alex99] Alexander, C.A.: The origins of pattern theory. The future of the theory and the generation of a living world. Keynote address recorded live at The 1996 ACM Conference on Object-Oriented Programs, Systems, Languages and Applications (OOPSLA). IEEE Software. September/October 1999

[BA01] Bergmans, L. And Aksit, M.: Advanced software composition: obstacles and approaches. Tutorial 16 ECOOP'01

[Beck00] Beck, K.: Extreme programming explained. Embrace change. Addison Wesley. 2000

[BMRS00] Buschman, F. et al: A system of patterns. Pattern Oriented software architecture. Wiley. 2000

[Bosc01] Bosch, J.: Design of Software architectures. ECOOP'01 Tutorial 04

[Busch01] Buschman, F.: Patterns at work. Tutorial ECOOP'01

[Butl01] Butler, G.: Objec-oriented frameworks. Tutorial 08 ECOOP'01

[Coco85] Cocorda-P,J: Les catastrophes de la parole de Roman Jakobson à René Thom. Collections Recherches Interdisciplinaires dirigée par Pierre Delattre. 1985

[Coco92] Cocorda-P, J.: Physique du sens. Éditions du Centre National de la Recherche Scientifique. 1992

[Copl00] Coplien, J.O.: Multi-paradigm design. PHD thesis. Vrje Universitet Brussel Informatics Department

[GC79] Greimas, A.J & Courtés, J.: Sémiotique: Dictionnaire raisonné de la théorie du langage. 2 vols. Paris: Hachette. Trad. 1982 Semiotics and Language. Bloomington: Indiana Univ. Press

[GHJV95] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design patterns – elements of reusable object-oriented software. Addison Wesley, Reading MA 1995

[Jahn01] Jahnke, J.H.: Engineering component-based net- centric systems for embedded applications. http://cs.uvic.ca/~jens or http://www.microcommander.com Proceedings

[Jahn01] Jahnke, J.H.: Engineering component-based net- centric systems for embedded applications. http://cs.uvic.ca/~jens or http://www.microcommander.com Proceedings of the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9) Vienna Austria, September 2001

[JÖ01] Jacobson, I. And Övergaard, G.: Making the software processes transparent by using intelligent agents. Tutorial 02 ECOOP'01

[KLMM97] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M. and Irwin, J.: Aspect-Oriented Programming. Keynote at ECOOP'97. Lecture Notes in Computer Science. Jyväskylä, Finland 1997

[KLMM97] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M. and Irwin, J.: Aspect-Oriented Programming. Keynote at ECOOP'97. Lecture Notes in Computer Science. Jyväskylä, Finland 1997

[Lore97] Lorenz, D.: Tiling design patterns. A case study using the Interpreter Pattern. Proceedings from OOPSLA'97, 206-217

[Lour00] Lourenci, A. O desdobramento e o modelamento da consciencia artistica na infoera. Relatorio Cientifico de pos-doutorado. FAPESP. http://www.lsi.usp.br/~lourenci

[Lour88] Lourenci, A. Espirito, energia e informacao: elementos essenciais do ecossistema urbano.Dissertacao de Mestrado. Departamento de Arquitetura e Urbanismo. EESC-USP. Junho de 1988.

[Lour97] Lourenci, A. The hermeneutic nature of an ecodesign model and Self, abstract, discussion inside the Programming language group in Position Papers from the 7th Workshop for PHD students in object-oriented systems Eds. F. Gerhardt, L. Wohlrab and E. Ernst published in ECOOP'97 Workshop Reader Lecture Notes in Computer Science 1357

[Lour98] Lourenci, A.: Uma proposta de um sistema de conhecimento orientado a objetos baseado em prototipos para a projetacao e planejamento de cidades sustentáveis. Tese de doutorado. FAU USP

[Maes87] Maes, P.: Computational reflection. Technical Report 87-2.1987

[Meye91] Meyer, B.: Introduction to the Theory of Programming Languages. Prentice Hall. 1991

[Nöth00] Nöth, W.: Semiotic Machines. Cybernetics and Human Knowing to appear in 8(2001) ou 9 (2002).

[Nöth98] Nöth, W. : Panorama da semiótica no século XX. Editora Anablume. 1998

[Nöth98] Nöth,W.:Panorama da Semiótica de Platão a Peirce. Editora Anablume

[Sant00] Santaella, L.: A teoria geral dos signos. Editora Pioneira. 2000

[SU96] Smith, R. and Ungar, D.:Us: A simple and unifying approach to subjective objects in Theory and Practice of object systems Vol. 2 (3), 161-178, 1996

[TDBL00] Tarr, P. , D'Hondt, M., Bergmans, L. and Lopes, C. V.: Workshop on Aspects and Dimensions of Concern: requirements on and challenge problems for advanced separation of concerns. ECOOP'00. http://trese.cs.utwente.nl/Workshops/adc2000. Published in Workshop Reader ECOOP'00

[YA00] Yacoub, S.M. and Ammar, H. H. Toward pattern-oriented frameworks. Journal of Object Oriented Programming January 2000

http://cs.uvic.ca/~jens or http://www.microcommander.com

RC00] Rayside, D. and Campbell, G.T.: An Aristotelian understanding of object-oriented programming. Proceedings from OOPSLA'00 337-353

References

[Abel00] Abel, C.: Architecture & identity. Responses to cultural and technological change. Architectural Press 2000

[AK00] Atkinson, C. and Kühne, T.: Separation of concerns through stratified architectures. Workshop on aspects and dimensions of concern: requirements on, and challenge problems for, advanced separation of concerns (orgs: Tarr, P, D'Hondt, M., Bergmans, L. and Lopes, C.V.) ECOOP 2000 Cannes France

[Alex99] Alexander, C.A.: The origins of pattern theory. The future of the theory and the generation of a living world. Keynote address recorded live at The 1996 ACM Conference on Object-Oriented Programs, Systems, Languages and Applications (OOPSLA). IEEE Software. September/October 1999

[AMT00] Aksit

[AMT00] Aksit, M., Marcelloni, F. and Tekinerdogan, B.: Developing object-oriented frameworks using domain models. ACM Computing Surveys Vol 32 March 2000

[AMT00] Aksit, M., Marceloni, F. and Tekinerdogan, B. Developing object-oriented frameworks using domain models. ACM Computing Surveys Vol. 32 March 2000

[AMT00] Aksit

[Apel81] Apel, K-O: Charles S Peirce From pragmatism to pragmaticismUniversity of Massachussets Press 1981

[Assu00] Assumpção, J. Mattos de: Design patterns and Self. Communication in the prototype based object oriented programming language Self list of discussion. E-mail from

[Assu00] Assumpção, J. M. de , Jr: Self/R. http://groups.yahoo.com/group/self-interest/links http://www.merlintec.com:8080/software Dec 13, 2001 Communication by e-mail.

[Assu01] Assumpcao, J. M., Jr.: Manager of the Self language group: self-interest@yahoo.com

[ATB] Aksit, M., Tekinerdogan, B. and Bergmans, L. Achieving adaptability through separation and composition of concerns. In Special Issues in Object Oriented Programming, M. Muhlhauser Ed, dpunkt Verlag pp.12-23 1996

[ATB]] Aksit, M., Tekinerdogan, B. and Bergmans, L. Achieving adaptability through separation and composition of concerns. In Special Issues in Object Oriented Programming, M. Muhlhauser Ed, dpunkt Verlag pp.12-23 1996

[BA01] Bergmans, L. and Aksit, M.: Advanced Software Composition: Obstacles and Approaches. Tutorial ECOOP'01 June 18-22, 2001 Budapest Hungary

[Barn89] Barnett,V. E. and Barnett, P. H. The originality of Kandinsky's compositions. The Visual Computer (1989) 5: 203-213

[Bäum00] Bäumer, D.: Software-architecturen für die rahmenwerkbasierte Konstruktion grosser Anwendungssysteme. Informatik der Universität Hamburg. PHD Thesis. 2000

[Berg00] Berge, B.: The ecology of building materials. Architectural Press. 2000

[BMRS00] A system of Patterns. Pattern-oriented software architecture. John Wiley . 2000

[Boeh00] Boehm, B: Unifying Software and Systems Engineering. IEEE Computer 33, 3 March 2000

[Brui98] Bruijn, S.G. de: Composable Objects with multiple views and layering. MSc. Thesis at the Department of Computer Science. University of Twente 1998

[Brui98]Bruijn, S.G. De Composable objects with multiple views and layering. MSC. Thesis at the Department of Computer Science SETI TRESE 1998

[Bruj98]]Bruijn, S.G. De Composable objects with multiple views and layering. MSC. Thesis at the Department of Computer Science SETI TRESE 1998

[Budd95] Budd, T.: Multi-paradigm programming in Leda. Reading, Ma: Addison-Wesley, 1995

[CD+93] Coleman, Derek et al: Object-oriented development: The Fusion Method.Engledwood Cliffs, NJ: Prentice-Hall. 1993

[Coc02] Cockburn, A.: Agile software development. http://members.aol.com/humansandt/crystal/game/getasddraft.htm Addison Wesley 2002.

[Cock01]] Cockburn, A.: Characterizing people as non-linear, first-order components in software development. http://www.CrystalMethodologies.org/ 2001

[Coco85] [Coco85] Cocorda-P,J: Les catastrophes de la parole de Roman Jakobson à René Thom. Collections Recherches Interdisciplinaires dirigée par Pierre Delattre. 1985

[Coco92] Cocorda-P, J.: Physique du sens. Éditions du Centre National de la Recherche Scientifique. 1992

[Cola87] Colapietro, V.: Is Peirce's general theory of the sign really general? In Transactions of the Charles Saunders Peirce Society. Spring 1987

[Copl01] Coplien, J.O.: *It's Time to Kill Software Engineering* Keynote at *XV SBES -Simpósio Brasileiro de Engenharia de Software; Primeira Conferência Latino-americana em Linguagens de Padrões para Programação e XVI SBBD- Simpósio Brasileiro de Banco de Dados* –October 1-5/2001, Rio de Janeiro

[Copl97] Coplien, J.O.: On the Nature of The Nature of Order. The Chicago Patterns Group presents James O. Coplien on Christopher Alexander's latest work entitled: The Nature of Order. In http://www.enteract.com/~bradapp/docs/NoNoO.html or http://www.rcnchicago.com/ ~**jcoplien**/bibliography.html

[Cosm00] Cosmides, L. and Tooby, J. Origins of domain specificity: the evolution of functional organization in Minds, brain and computers. Editors: Cummins, R. and Cummins, D.D. Blackwell Publishers 2000

[CZ00] Coplien, J. O. and Zao, L.: Symmetry and symmetry breaking in software patterns. Proceedings of the Second International Workshop on Generative Components Programming. 2000

[Dear00] Dear, M.J.: The postmodern urban condition. Blockwell Publishers. 2000

[Dijk89] Djikstra, E.: On the cruelty of really teaching computer science. CACM 32, 12 December 1989

[DLC93] Duffy, F.; Laing, A. and Crisp, V.: The responsible workplace. The redesign of work and offices. Butterworth Architecture in association with Estates Gazette. 1993

[Edwa99] Edwards, B.: Sustainable architecture. European directives & building design. Architectural Press. 1999

[EXPO00] Fitas gravadas semanalmente sobre a EXPO2000: Homem, Natureza, Técnica. Deutsche Welle.

[Ferg92] Ferguson, E.S.: Engineering and the Mind's Eye MIT Press 1992

[Foo, J]: Integrated bio-systems: a global perspective. Integrated Bio-Systems Network InternationalOrganization on Biotechnology and Bioengineering http://www.ias.unu.edu/proceedings/icibs/ibs/ibsnet/InFoRM2000-paper.htm

[Fowl01] Fowler, M.: The new methodology. http://www.martinfowler.com/articles/newMethodology.html

[Gada75] Gadamer, H.G. Truth and Method. Sheed and Ward. 1975

[GC79] Greimas, A.J & Courtés, J.: Sémiotique: Dictionnaire raisonné de la théorie du langage. 2 vols. Paris: Hachette. Trad. 1982 Semiotics and Language. Bloomington: Indiana Univ. Press

[GHJV95] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design patterns – elements of reusable object-oriented software. Addison Wesley, Reading MA 1995

[GL96] Gössel, P. and Leuthäuser, G.: Arquitectura no século XX. Taschen 1996

[HH90] Helm, R. Holland, I. And Gangopadhyay: Contracts: specifying compositions in object-oriented systems ECOOP/OOPSLA'90 Proceedings

[High01] Highsmith, J.: *Is "Software Engineering" the Wrong Metaphor? And Why Should We Care?* at *OOPSLA 2001- ACM Conference on Object Oriented Programming Systems, Languages and Applications*, October 14-18, 2001, Tampa Florida)

[Holl92] Holland, I. Specifying reusable components using contracts. Proc. Of the ECOOP'92 Conference. LNCS 615 Springer Verlag 1992 pp. 287-308

[HS00] Hughes, J. and Sadler, S.: Non-plan. Essays on freedom participation and change in modern architecture and urbanism. Architectural Press 2000

[Jack01] Jackson , M.: *Where, Exactly, is Software Engineering?* Keynote at *Joint 8th European Software Engineering Conference (**ESEC**) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering **(FSE-9)** held in Vienna September 10-14/2001*

[Jack01] Jackson, M.: Where, exactly, is Software Engineering? Keynote at the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9) (Ed. By Volker Gruhn. Vienna, Austria September 10-14, 2001

[Jahn01] Jahnke, J.H.: Engineering component-based net- centric systems for embedded applications. Proceedings of the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9) Vienna Austria, September 2001http://cs.uvic.ca/~jens or http://www.microcommander.com

[JJ95] Jellicoe, G. and S.: The landscape of man. Shaping the environment from prehistory to the present day. Thames and Hudson. 1995

[Jone 98] Jones, D.L.: Architecture and environment. Bioclimatic building design. Laurence King 1994

[Kata01] Katayama, T.: Evolutionary domains: a basis for sound software evolution in Proceedings from IVth International Workshop on Principles of Software Evolution. Vienna September 10-14/2001

[Kicz91] Kiczales, G. et al: The Art of the Metaobject Protocol MIT Press. 1991

[KLMM97] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M. and Irwin, J.: Aspect-Oriented Programming. Keynote at ECOOP'97. Lecture Notes in Computer Science. Jyväskylä, Finland 1997

[Knie00] Kniesel, G.: Dynamic object-based inheritance with subtyping. PHD thesis. University of Bonn. 2000

[Knie00] Kniesel, G.: Dynamic Object-based Inheritance with Subtyping PhD thesis. Bonn University. Informatics Institut III July 2000

[Knie00] Kniesel, G.: Dynamic object-based inheritance with subtyping. PHD thesis. University of Bonn. 2000

[Koest64] Koestler, A.: The act of creation . Macmillan. 1964

[Kuhn70] Kuhn, T.: *Structure of Scientific Revolutions*. Chicago: University of Chicago Press, 1990

[Kuhn70] Kuhn, T.: *Structure of Scientific Revolutions*. Chicago: University of Chicago Press, 1990

[Lewi79] Lewis, F. P.: History of the parallel postulate. In Selected papers on Geometry. Edited by Stehney, A. K. et al. The Mathematical Association of America. 1979

[Lönn89] Lönnrot, E.: The Kalevala. Oxford University Press 1989

[Lore97] Lorenz, D.: Tiling design patterns. A case study using the Interpreter Pattern. Proceedings from OOPSLA'97, 206-217

[Lour00] Lourenci, A. O desdobramento e o modelamento da consciencia artistica na infoera. Relatorio Cientifico de pos-doutorado. FAPESP. http://www.lsi.usp.br/~lourenci

[Lour88] Lourenci, A. Espirito, energia e informacao: elementos essenciais do ecossistema urbano.Dissertacao de Mestrado. Departamento de Arquitetura e Urbanismo. EESC-USP. Junho de 1988.

[Lour97] Lourenci, A. The hermeneutic nature of an ecodesign model and Self, abstract, discussion inside the Programming language group in Position Papers from the 7th Workshop for PHD students in object-oriented systems Eds. F. Gerhardt, L. Wohlrab and E. Ernst published in ECOOP'97

[Lour98] Lourenci, A.: Uma proposta de um sistema de conhecimento orientado a objetos baseado em prototipos para a projetacao e planejamento de cidades sustentáveis. Tese de doutorado. FAU USP

[LR00] Lehman, M.M. and Ramil, J.F.: Software evolution in the age of component based software engineering. Software Engineering. 2000

[LR01] Lehman, M.M. and Ramil, J.F.: Software Evolution. *Proceedings from International Workshop on Principles of Software Evolution.pp. 1-13*

[LR01a] Lehman, M.M. : Communications by e-mail.October 19.

[LR01b] Lehman, M.M. and Ramil, J.F.: An approach to a theory of software evolution. Proceedings from International Workshop on Principles of Software Evolution pp. 62-64

[Mack97] Mackenzie, D.: Green design. Design for the environment. Laurence King. 1997

[Maes87] Maes, P.: Computational Reflection. Artificial Intelligence Laboratory. Vrie University Brussels. 1987

[Maso64] Mason, C.E.: Rainer Maria Rilke. Sein Leben und sein Werk. Vandenhoeck & Rupecht in Goettinge, 1964

[Mele99] Melet, Ed.: Sustainable architecture. Towards a diverse built environment. NAI Publishers. 1999

[Mel-K95] Machado, R. And el-Khoury, r.: Monolithic architecture. Prestel 1995

[MG01] Mens, T. and Galal, H. G.:

[Mitt01] Mittermeier, R.: Sharpening the terminology before sharpening the tools. Proceedings from International Workshop on Principles of Software Evolution

[Mune97] Muneer, T.: Solar radiation & daylight models for the energy efficient design of buildings. Architectural Press 1997

[MV80] Maturana, H. and Varela, F. Autopoiesis and cognition: the realization of the living. Dordrecht: D. Reidel, 1980

[Naur85] in Agile software development. Alistair Cockburn. Addison Wesley 2002

[NB96] Norris, C. & Benjamin, A.: *What is deconstruction?* Academy editions. 1996

[Nerd99] Nerdinger, W. (ed.): Alvar Aalto. Toward a human modernism. Prestel

[Noev97] Noever, P.*: Architecture in transition*. Between deconstruction and new modernism. Prestel 1997

[Norb88] Norberg-Schulz, C.: Roots of modern architecture. A.D.A. EDITA 1988

[Nöth02] Nöth, W.: Semiotic Machines. Cybernetics and Human Knowing to appear in 8(2001) ou 9 (2002).

[Nöth98] Nöth, W. : Panorama da semiótica no século XX. Editora Anablume. 1998

[Nöth98b] Nöth,W.:Panorama da Semiótica de Platão a Peirce. Editora Anablume

[OAO98] Otterpohl, R., Albold, A. and Oldenburg, M.: Differentiating management resource of water and waste in urban areas. Proceedings of the Internet Conference on Integrated Bio-Systems (1998) promoted by Institute of Advanced Studies, UN University and Microbial Resources Centres UNESCO http://www.ias.unu.edu/proceedings/icibs

[OT99] Ossher, H. and Tarr, P.: Multi-dimensional separation of concerns and the Hyperspace approach. IBM Research Report 21 452 April 1999

[Papa95] Papanek, V.: Arquitectura e Design. Edições 70. 1995

[Patt96] Patte, H.H.: The physics of symbols and the evolution of semiotic controls. Paper presented at the Workshop on Control Mechanisms for Complex Systems: Issues of Measurement and Semiotic Analysis. Las Cruces, New Mexico, Dec. 8-12, 1996. *Proceedings from Santa Fe Institute Studies in the Sciences of Complexity* Addison Wesley, 1997

[Prig62] Prigogine, I. Introduction to non-equilibrium thermodynamics. Wiley 1962

[PWCC] Paulk, M., Weber, C., Curtis, B., and Chrissis, M.B.: The capability maturity model. Addison Wesley 1995

[RC00] Rayside, D. and Campbell, G.T.: An Aristotelian understanding of object-oriented programming. Proceedings from OOPSLA'00 337-353

[Ross59] Ross, W. D.: The works of Aristotle. The Clarendon Press. 1959

[Ruan99] Ruano, M.: Ecourbanismo. Entornos humanos sostenibles: 60 proyectos. Ecourbanism. Sustainable human settlements: 60 case studies. Gustavo Gili 1999

[SAAH+] Schreiber, G.; Akkermans, H.; Anjewierden,A., Hoog, R. de, Shadbolt, N.,Velde, W. van de, Wielinga, B.: Knowledge Engineering and Management: The CommonKADS Methodology. MIT Press Cambridge MA 2000 ISBN 0-262-19300-0

[Sarf00] Post-quantum mechanics. http://www.well.com/user/sarfatti/index.html

[Schw74] Schwarzenberger, R.L.E.: The 17 plane symmetry groups. Mathematical Gazette LXIII, pp. 123-131

[SCT01] Shull, F., Carver, J. And Travassos, G.H.: An empirical methodology for introducing software processes. Proceedings of the Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9) Vienna Austria, September 2001

[SL01] Sreedhar, V.C. and Lorenz, D. orgs: First OOPSLA Workshop on language mechanisms for programming software components Tampa, Florida October 14-18/2001

[Stey94] [Stey94] Steyaert, P.: *A foundation for specialisable reflective language frameworks*. Vrije Universiteit Brussel Departement Informatica PhD Thesis1994

[SU96] Smith, R. and Ungar, D.:Us: A simple and unifying approach to subjective objects in Theory and Practice of object systems Vol. 2 (3), 161-178, 1996

[TDBL00] Tarr, P. , D'Hondt, M., Bergmans, L. and Lopes, C. V.: Workshop on Aspects and Dimensions of Concern: requirements on and challenge problems for advanced separation of concerns. ECOOP'00. http://trese.cs.utwente.nl/Workshops/adc2000. Published in Workshop Reader ECOOP'00

[TO93] Tarr, P. and Ossher, H.: Subject-oriented programming (a Critique of pure objects) in OOPSLA'93

[TO96] Tarr,P. and Ossher, H.: Subject-oriented programming and design patterns. http://www.research.ibm.com/sop/sopcpats.html

[Triv89] Trivedi, K. Hindu temples: models of a fractal universe. The Visual Computer (1989) 5: 243-258

[Unga95] Ungar, D.: *How to program Self 4.0*. Sun Microsystems laboratory (pp.1-102)

[VV00] Vale, B. And R.: The new autonomous house. Design and planning for sustainability. Thames and Hudson. 2000

[YA00] Yacoub, S.M. and Ammar, H. H. Toward pattern-oriented frameworks. Journal of Object Oriented Programming January 2000