Integral Systems Engineering Methodology

Kent Palmer

2011.04.11

http://kdp.me kent@palmer.name

Problematic:

What is the foundation of Systems Engineering?

Answer: Systems Theory => Schemas Theory

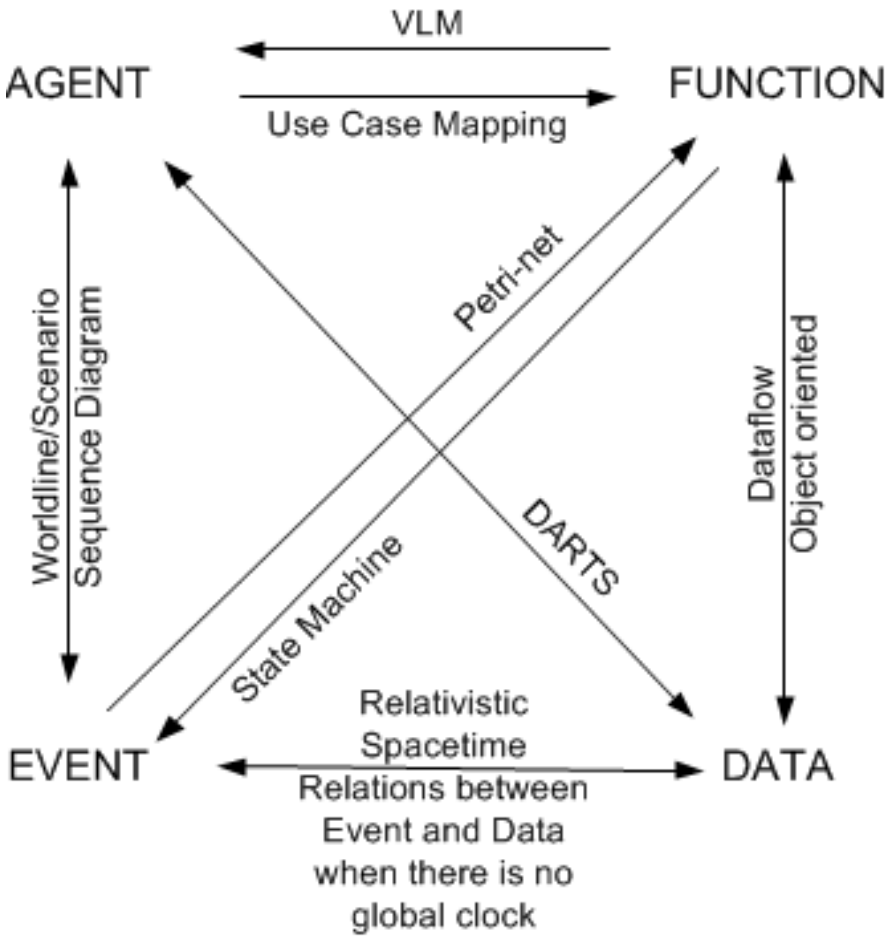
Given that, what is the nature of Design?

Answer: Quadralectics

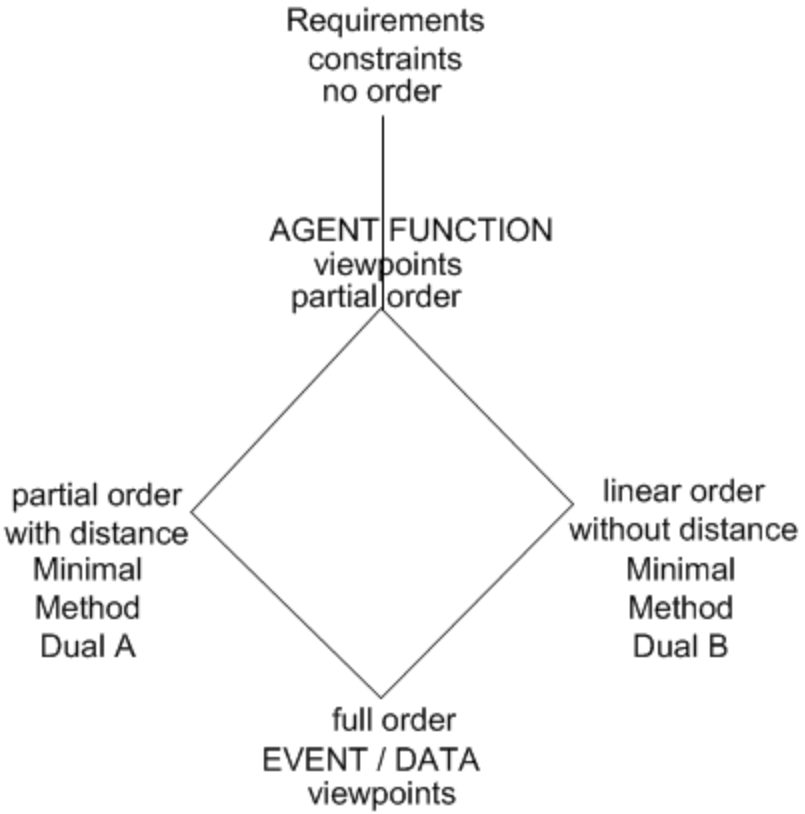
Given that, what is the implication for Practice?

Answer: Language Oriented Design

Research Program began with an attempt to understanding Real-time Software Architectural Design Methods

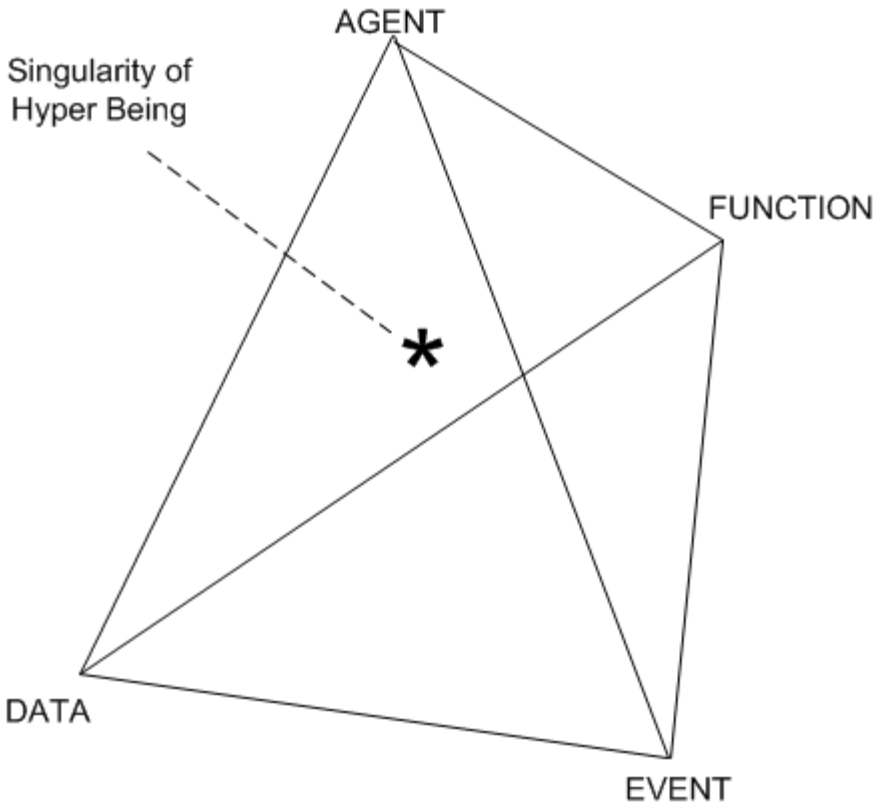


Methodological Distinctions



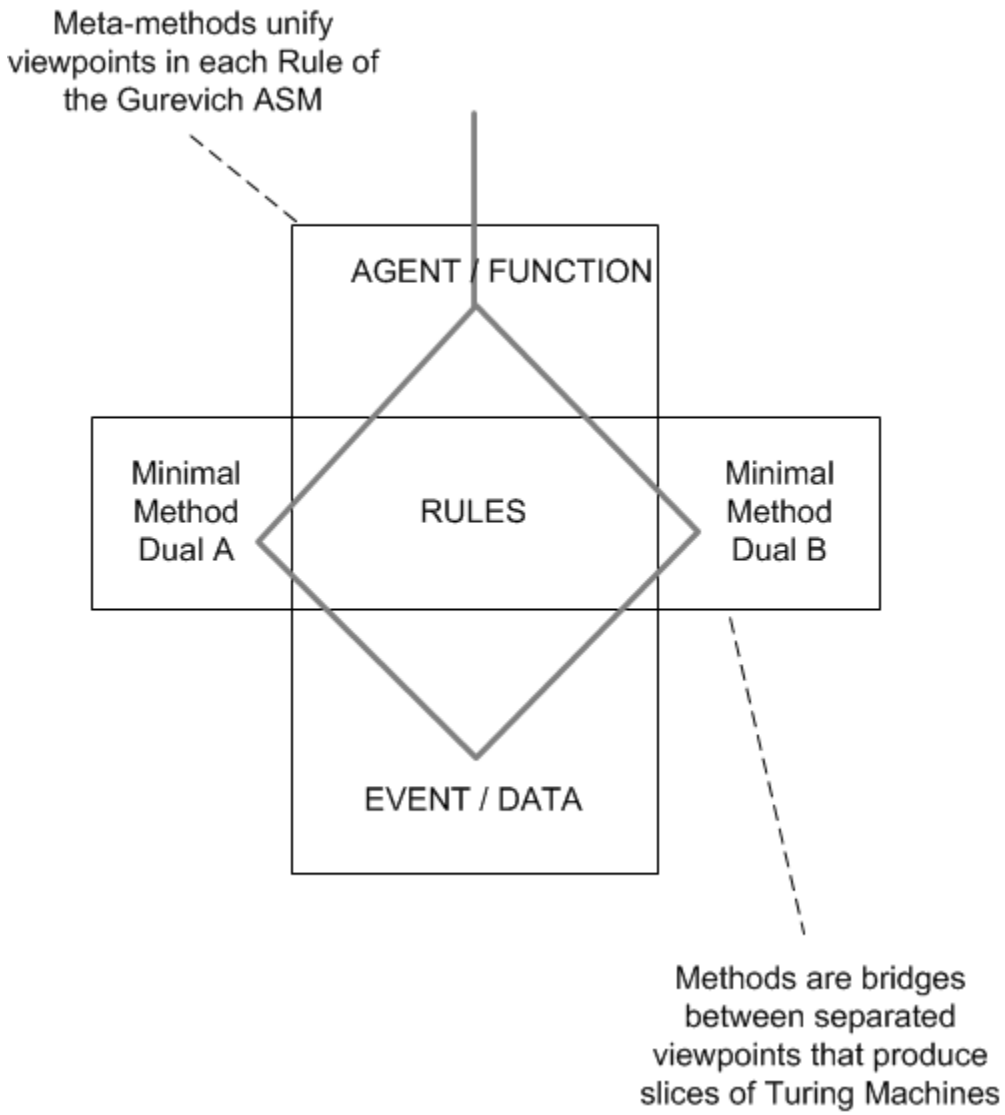
G. Klir's Methodological Distinctions and the relationships between the Viewpoints and Minimal Method Duals

Software and Hyper Being



Hyper Being singularity at core of Real-time System

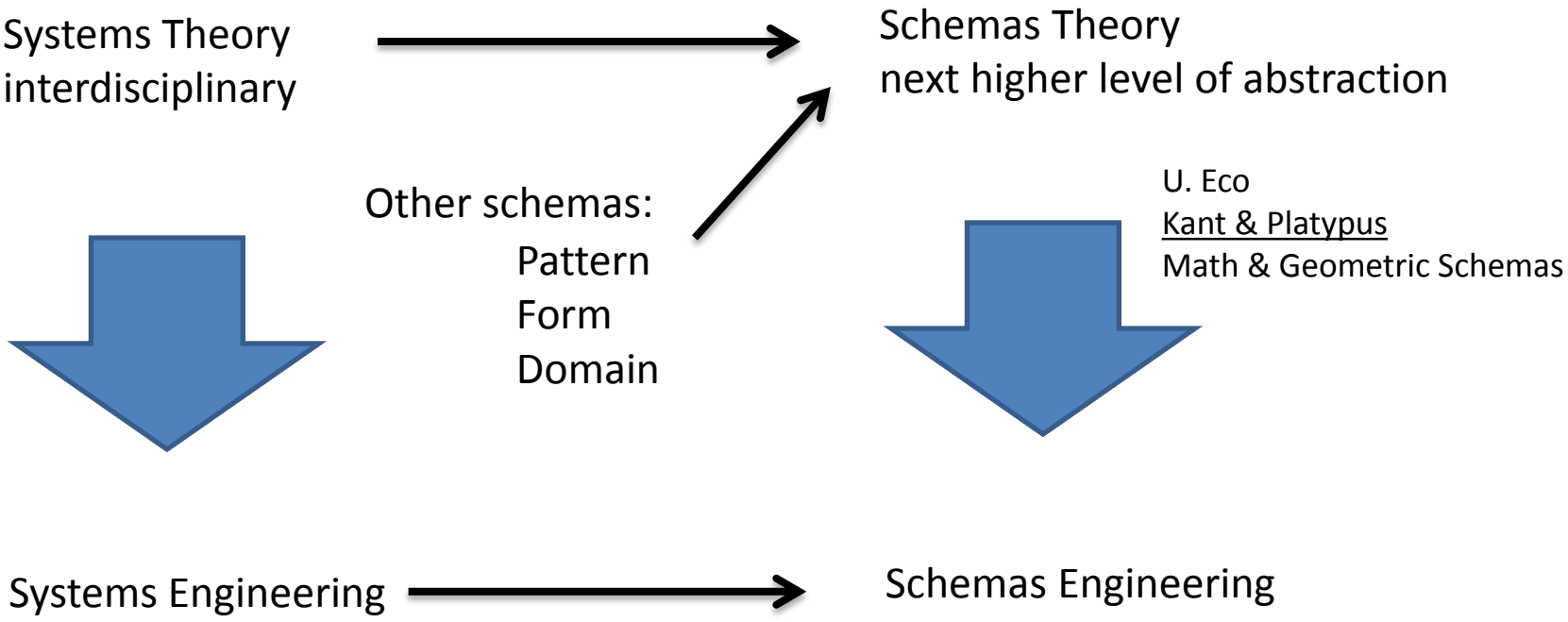
GASM and Minimal Methods of Design are duals



ISEM

- Four viewpoints on Real-time System Design
- Minimal methods are slices of a Turning machine and that explains the wholeness of the methodological field
- Viewpoints related to G. Klir ASPS Methodological Distinctions causes Field to be lopsided
- Fits into Klir's concept of Background Variables
  - Population, Space, Time
- Many of the minimal methods appear in UML/SYSML profiles
- Each minimal method can be expressed in its own Architectural Domain Specific Language (ADSL)
- See Wild Software Meta-systems circa 1996

Transitioned from Software Methods research to working on Systems Engineering Foundations



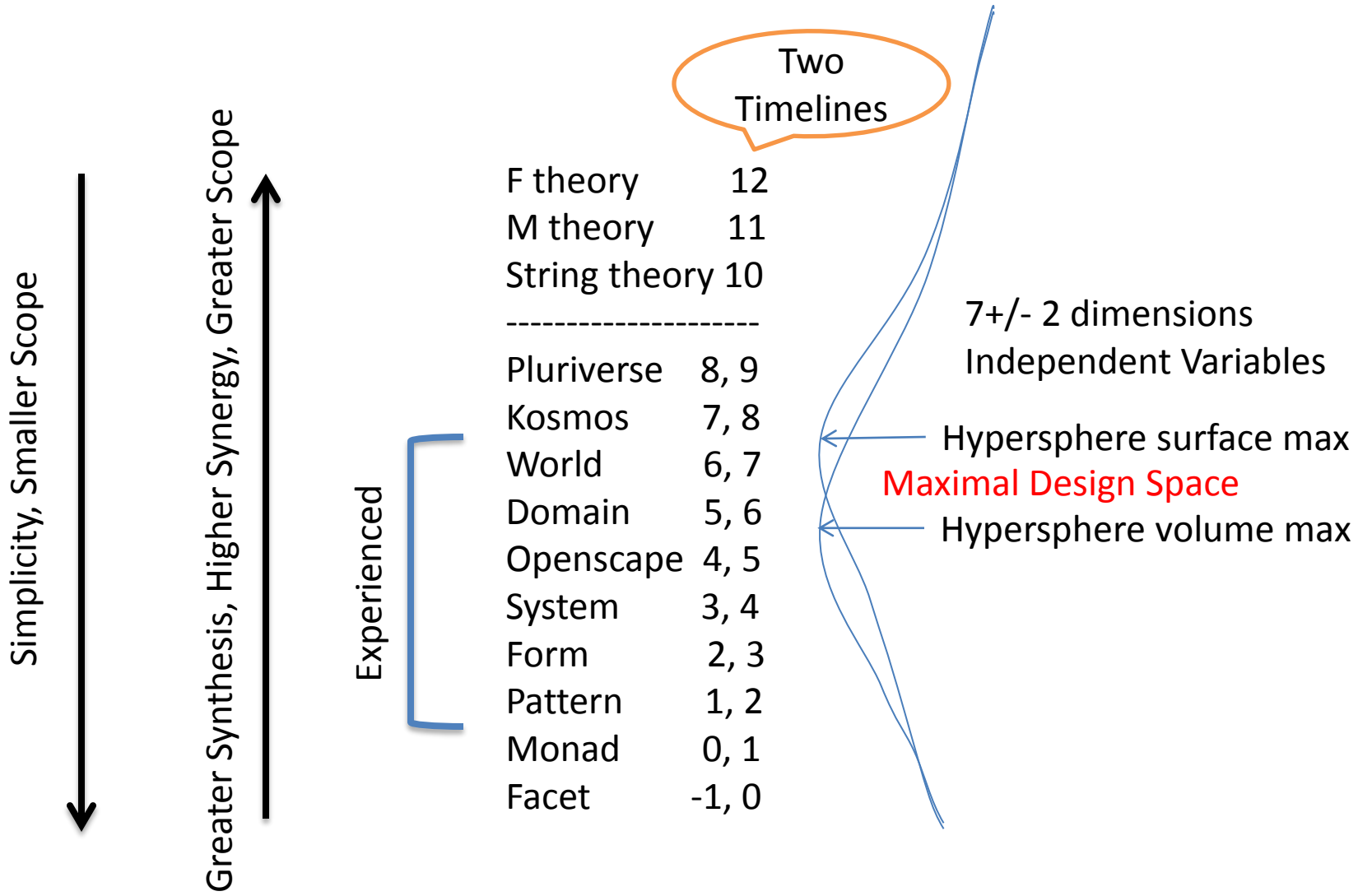
Hypothesis ***S'***

Ten Schemas

	dimensions
Pluriverse	8, 9
Kosmos	7,8
World	6, 7
Domain	5, 6
OpenScape	4, 5
System	3,4
Form	2, 3
Pattern	1, 2
Monad	0, 1
Facet	-1, 0

- Schemas are the templates of understanding spacetime organizations, which we project a priori (ala Kant)
- Rule:
  - Two Schemas per Dimension
  - Two Dimensions per Schema
- Nested Hierarchy with Different Scopes
- Autopoietic Reflexive Structure

Natural Limits of our Comprehension of Organizational Structures of Phenomena



Design as Sign Engineering

Pieter Wisse in his dissertation pointed out the semiotic dimension of engineering

Meta-levels of the Sign
Design is the third meta-level of the sign

Kind of Being	Kind of sign	Mode of being-in-the-world	Psychological concomitant	ego type
Ultra	obsign	handless	lost	singularity
Wild	resign	out-of-hand	encompass	enigma
Hyper	design	in-hand	bear	query
Process	ensign	ready-to-hand	grasp	Dasein/eject
Pure	sign	present-at-hand	point	Subject/object

Design:

- Differ*a*nce (differing/deferring)
  - Heidegger ~~Being~~ (crossed out)
  - Derrida
  - “slip-sliding away” (Paul Simon)
- Grammatology (science of traces)
- Traces of potentials in possibility
  - Lead to emergent eventities
- Plato’s Third type of Being in the Timeous

Differ*a*nce: non-representablity

- Peter Naur
  - No amount of documentation can capture a design completely, you must talk to the designer to understand a design completely
- Software as an artifact has the nature of Differance embodied within it
- Software makes systems adaptable but also makes design very difficult
- Design is fated to embody non-representable differance

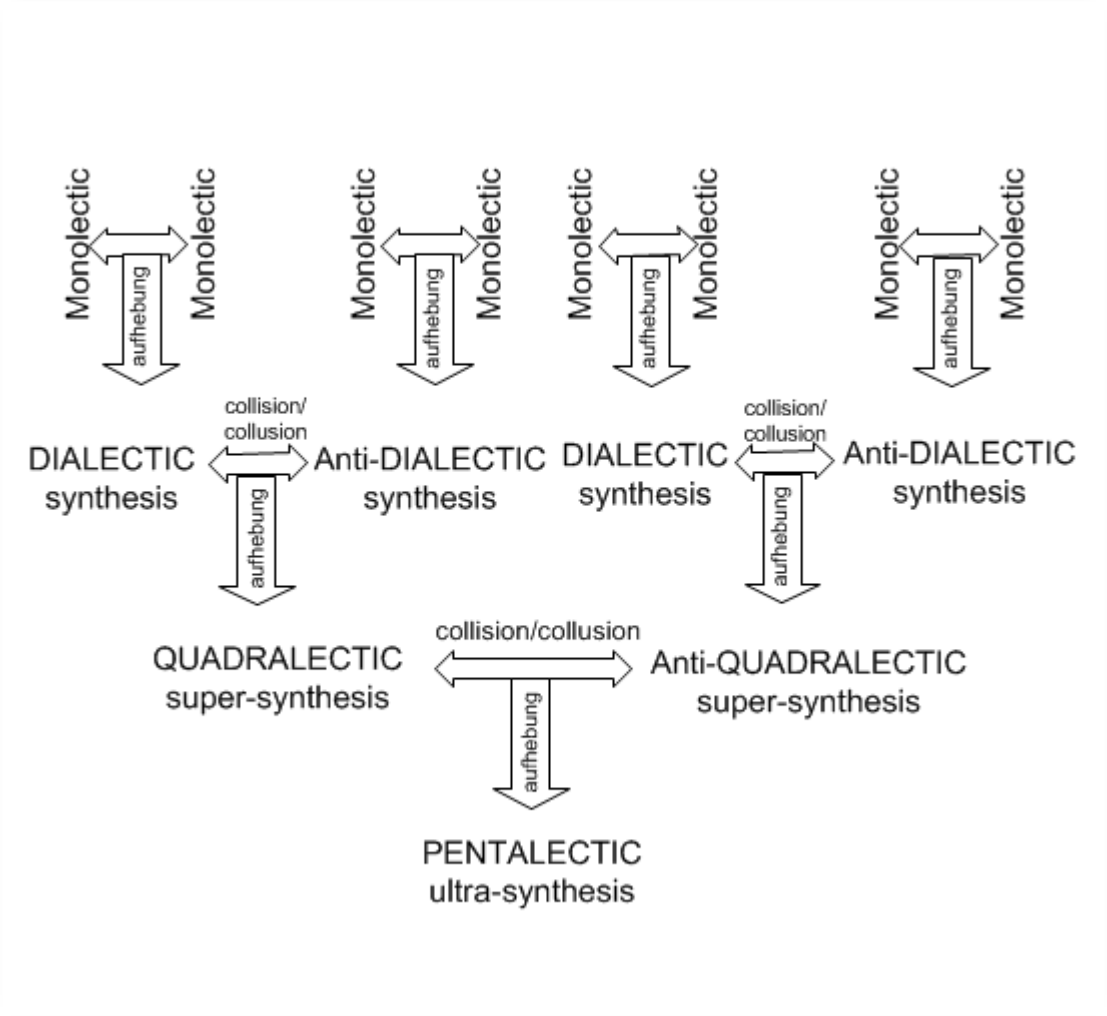
How design embodies non-representable differ*a*nce

- 0d - Monolectic - Dogmatic Uncritical Philosophy
- 1d - Dialectic - KANT static, HEGEL dynamic
- 2d - Trialectic - HEGEL work (prior to advent of spirit)
- ----------
- 3d - Quadralectic – B. Fuller dynamic minimal system
- 4d - Pentalectic – Synergistic system and meta-system

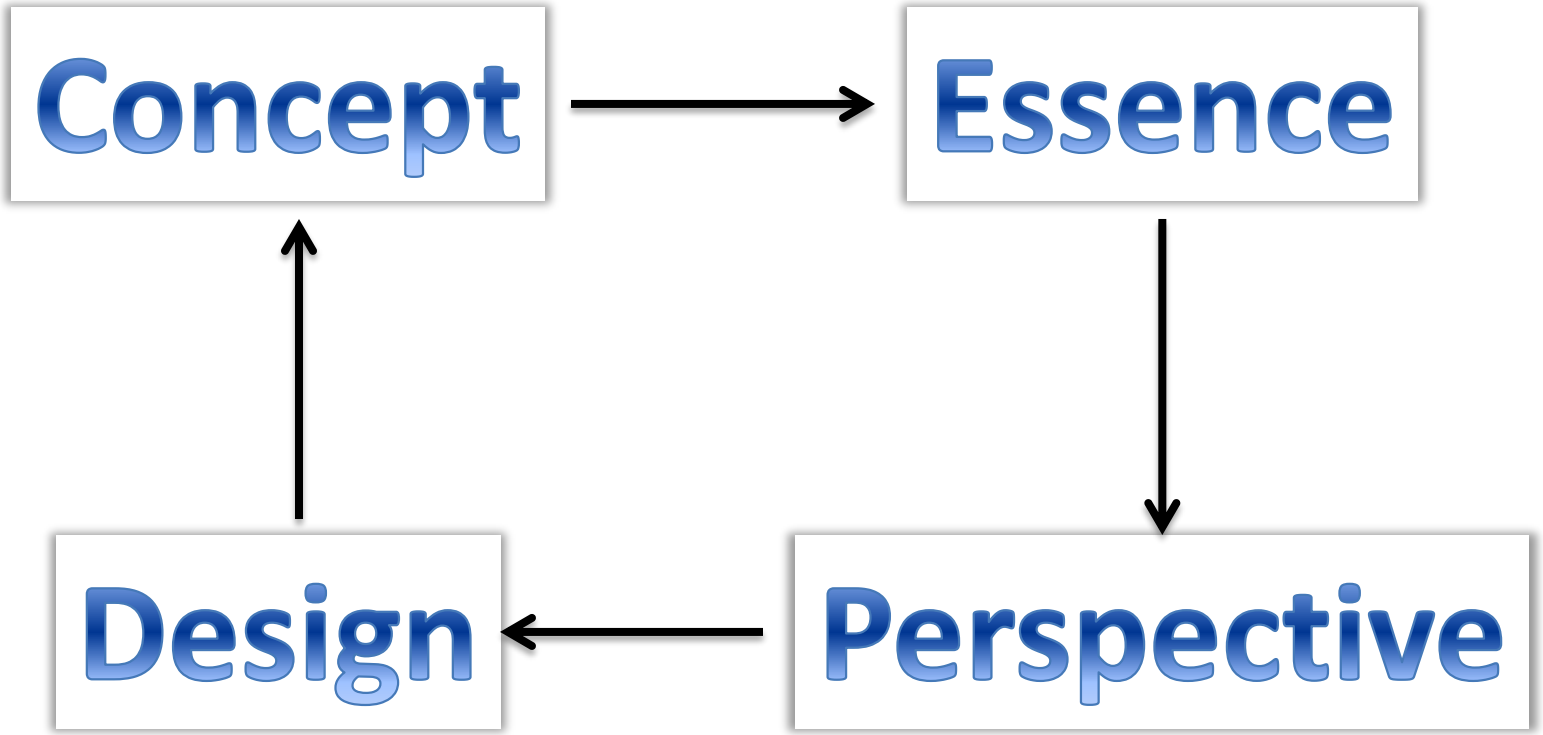
Multilectics Structured by Meta-levels of Being

Nature	Multilectic	Kind of Being	Characteristic	Concommitant
Dogmatism of the uncritical 0d	Monolectic	Pure	Static	Point
Aufhebung 1d	Dialectic	Process	Dynamic	Grasp
Work that gives rise to the product 2d	Trialectic	Hyper	Slippery	Bear
Minimal System 3d (tetra, knot, torus, mobius)	Quadralectic	Wild	Fragmented	Encompass
Pentachora in 4d space	Pentalectic	Ultra	Singularity	Lost

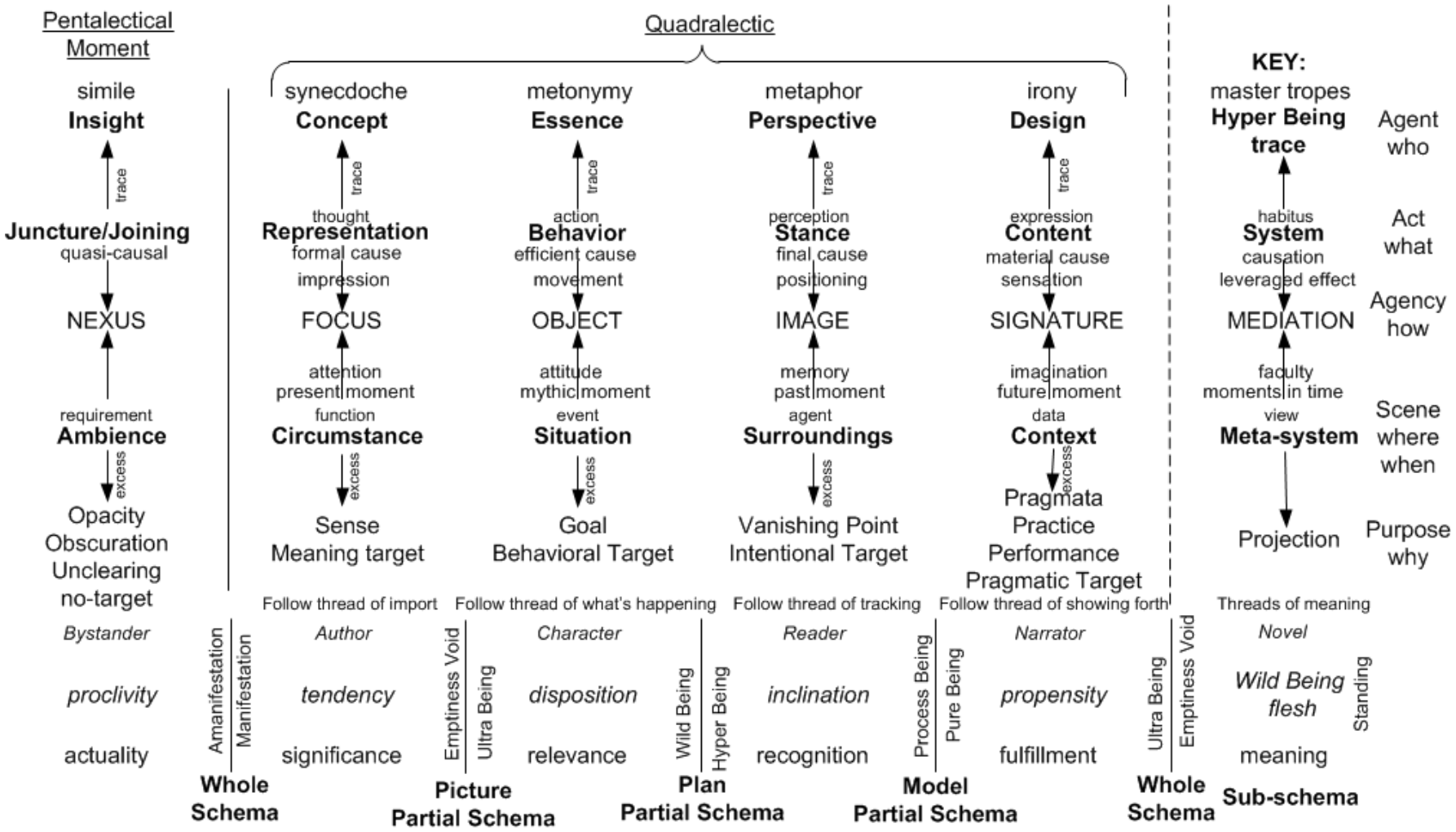
Composition of the Quadralectic



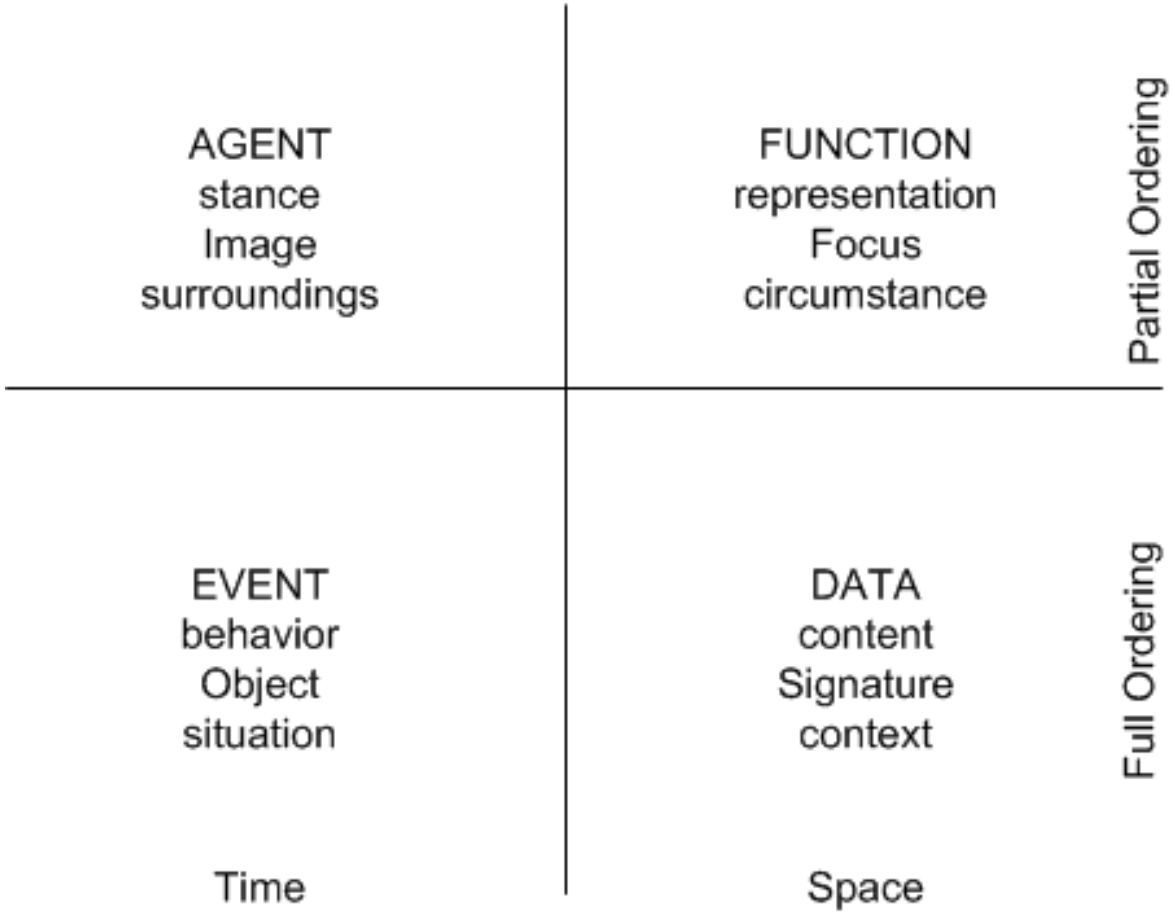
Quadralectics of Design:
Moments at non-representable trace level



Moments of the Quadralectic



Four Design Viewpoints and Quadralectic



Peirce's Principles of the Architectonic

Principle	Characteristic	Source
Seventh	Outside the singularity	new
Sixth	Poise	new
Fifth	Integrity	Fuller
Fourth	Synergy	Fuller
Third	Synthesis (Continuity)	Peirce
Second	Relata	Peirce
First	Isolata	Peirce
Zeroth	Void/Emptiness	new
Neganary	Inside the singularity	new

Design Field is widest at the Hyper Being level
Quadralectic is inscribed in the Design Field

	First	Second	Third	Fourth
Ultra	Subtlety	Affinity	Mutable	Simultaneity
Wild	Refinement	Inflection	Malleable	Ensemble
Hyper	Qualia	Kind	Topology	Synergy
Process	Spectra	Category	Manifold	Lattice
Pure	Property	Map	Continuity	Interdepen-dence
being	individual	element	array	component

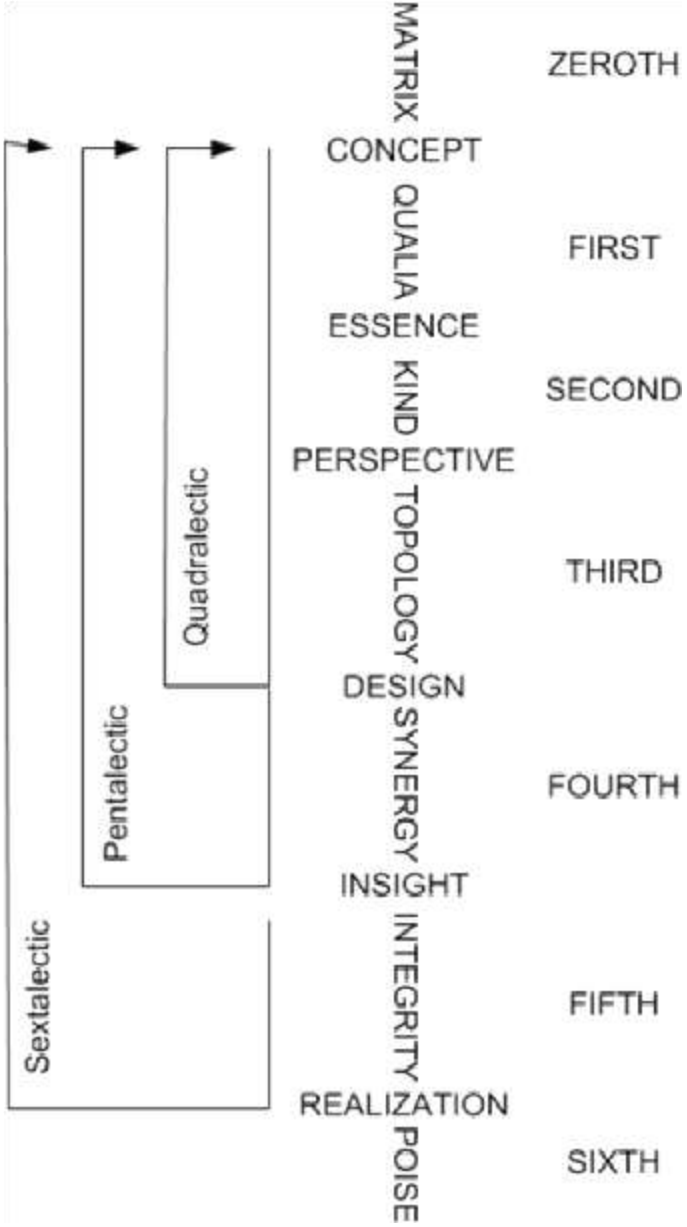
Concept

Essence

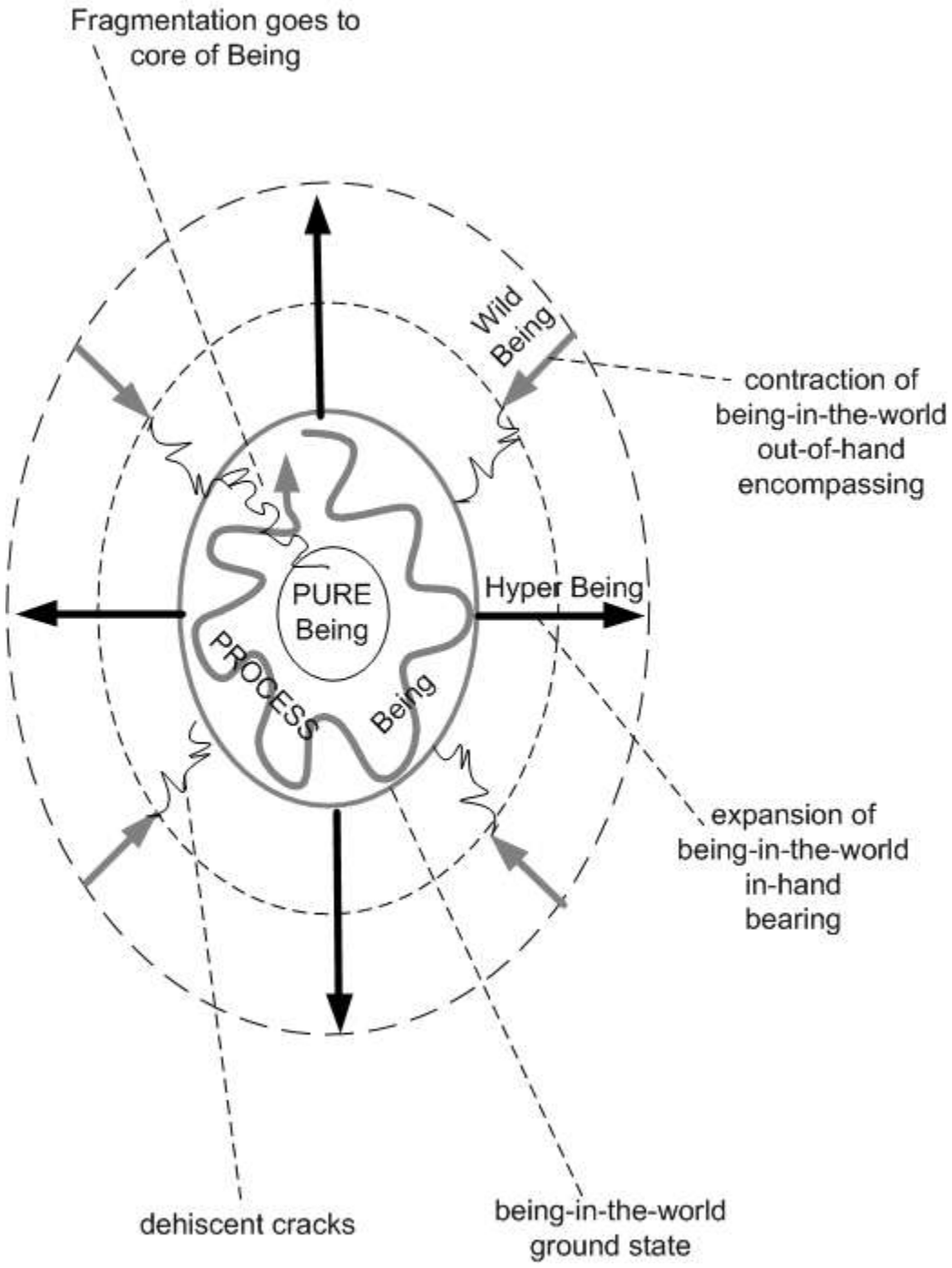
Perspective

Design

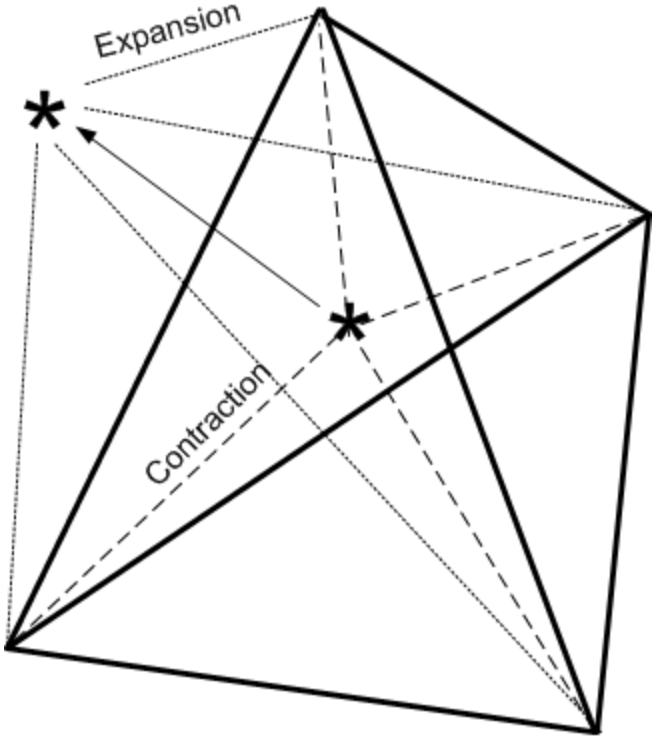
Design appears as we move the synthesis toward higher levels of synergy, integrity, and poise



Wild Being and Hyper Being are duals of each other related to the contraction and expansion of being-in-the-world



Quadralectic to Pentalectic



Pentahedron contracted embedding or expansion into the fourth dimension

Self-Dual
Pentahedron
1-5-10-10-5-1
V E F S

A^5

Order 60

Icosahedron

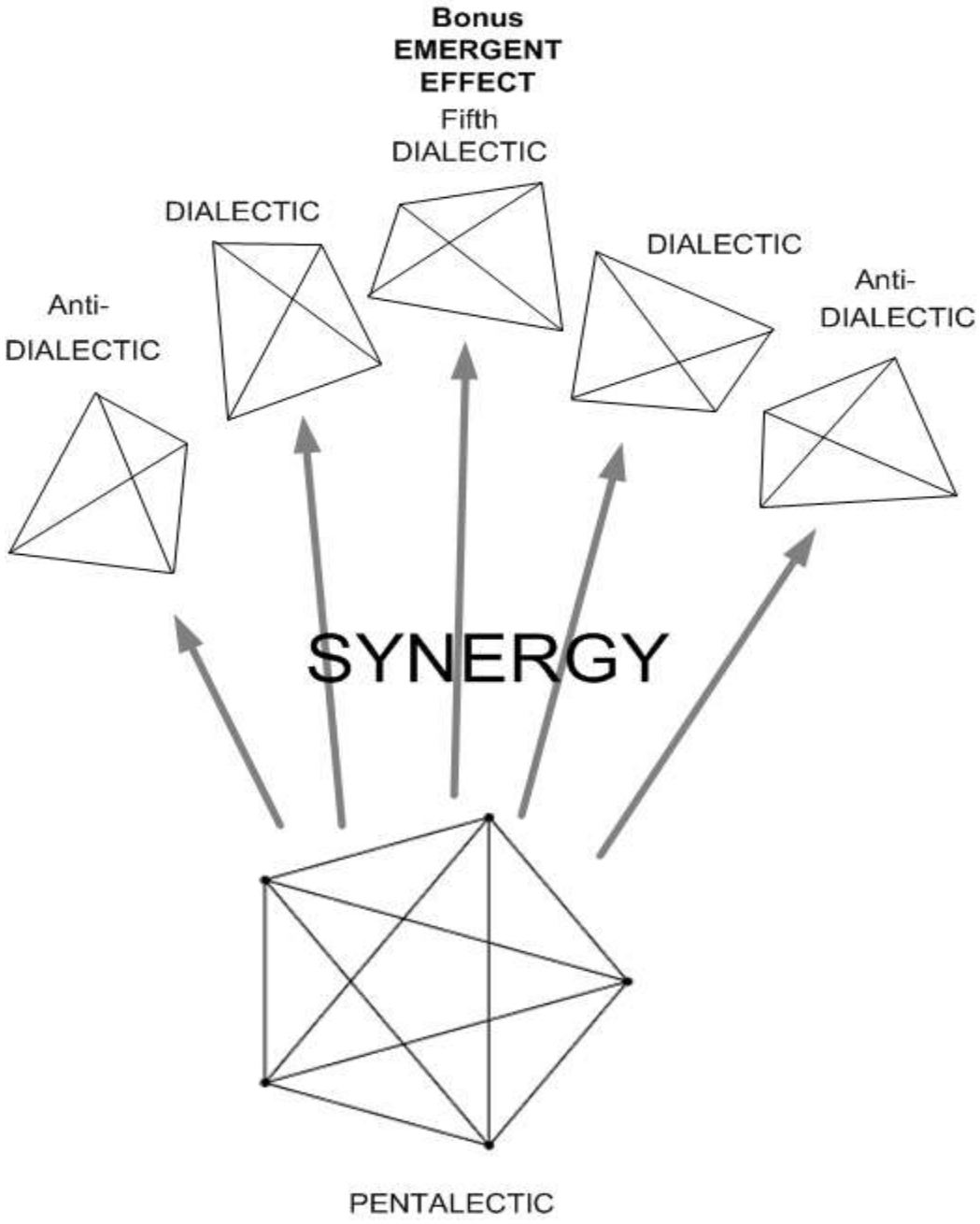
1-12-30-20-1
V E F

DUALITY

V E F
1-20-30-12-1

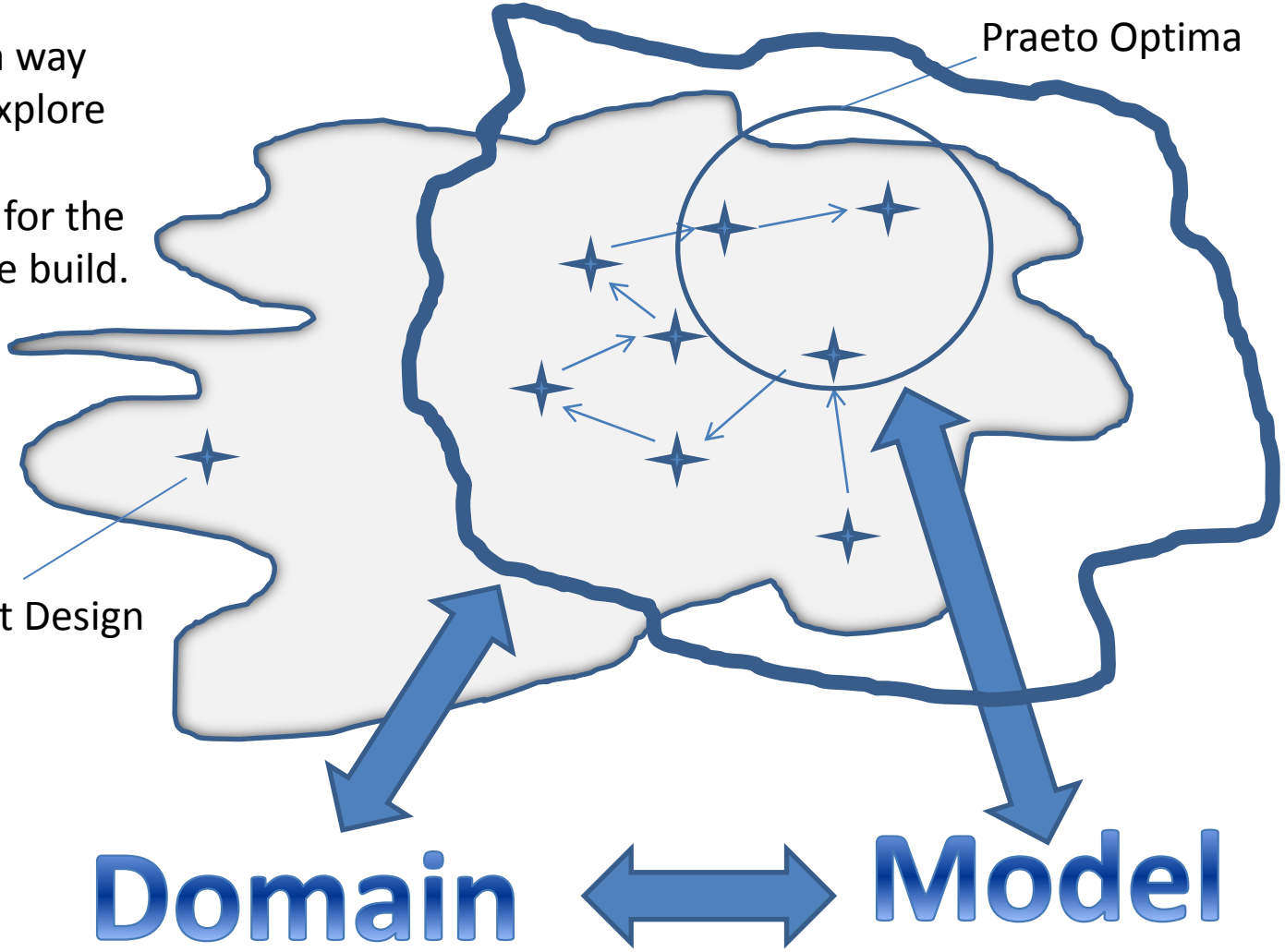
Dodecahedron

Pentalectic has an emergent surplus signified by the fifth Tetrahedron



Design Landscape

We need a way to easily explore the design landscape for the systems we build.



Answer

Domain Specific Languages
that incorporate Models
which can be varied
to move through the design landscape
and represent domain specific concepts
as well as the minimal methods used in the architectural domain of System Design

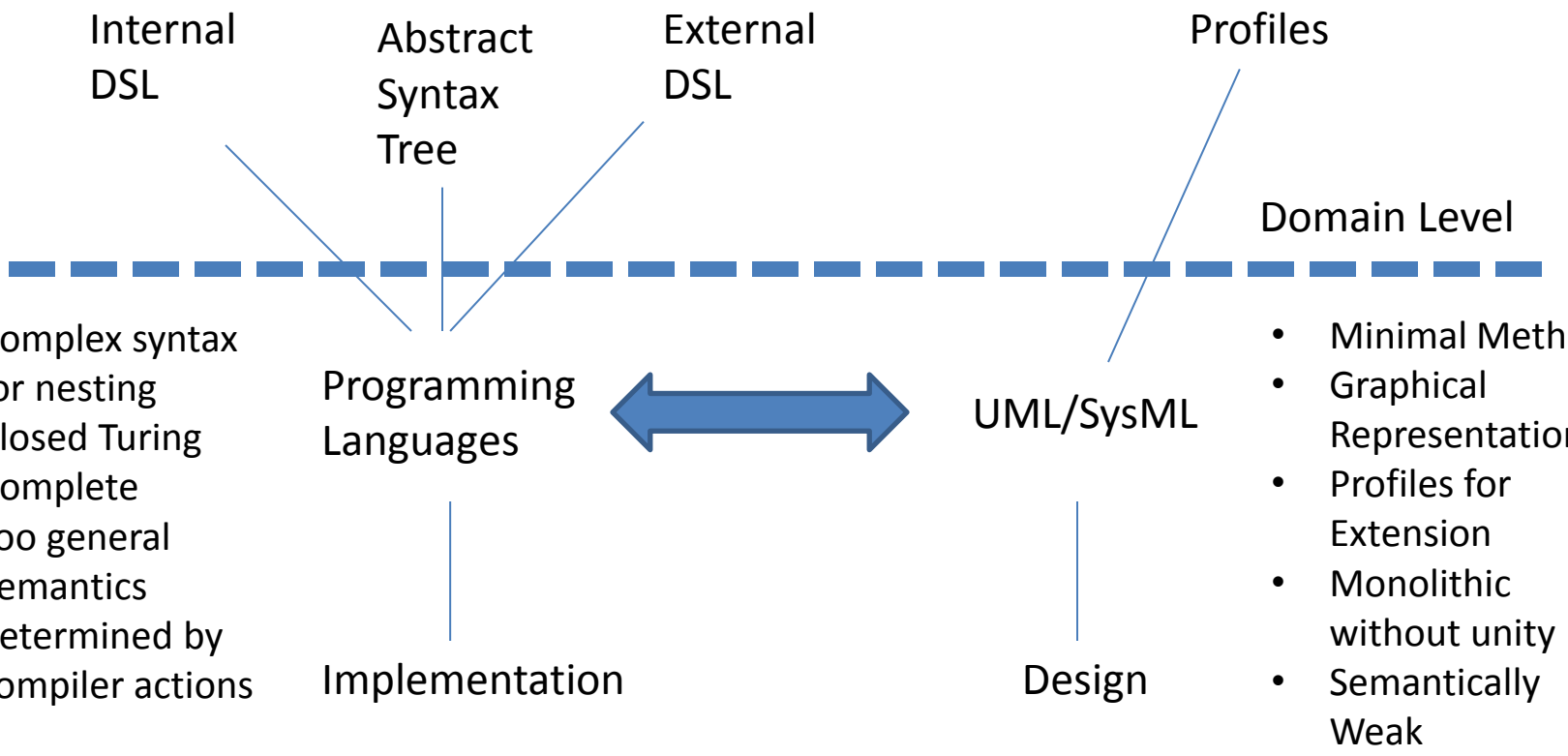
DSL issues

- Conforms to language syntax and semantics
- Not understood by domain experts
- Code

- Difficult to work with
- Closer to Model

- Textual
- Parser necessary
- Non-standard
- Technologically Intensive

- Non-standard must be developed and added to tool



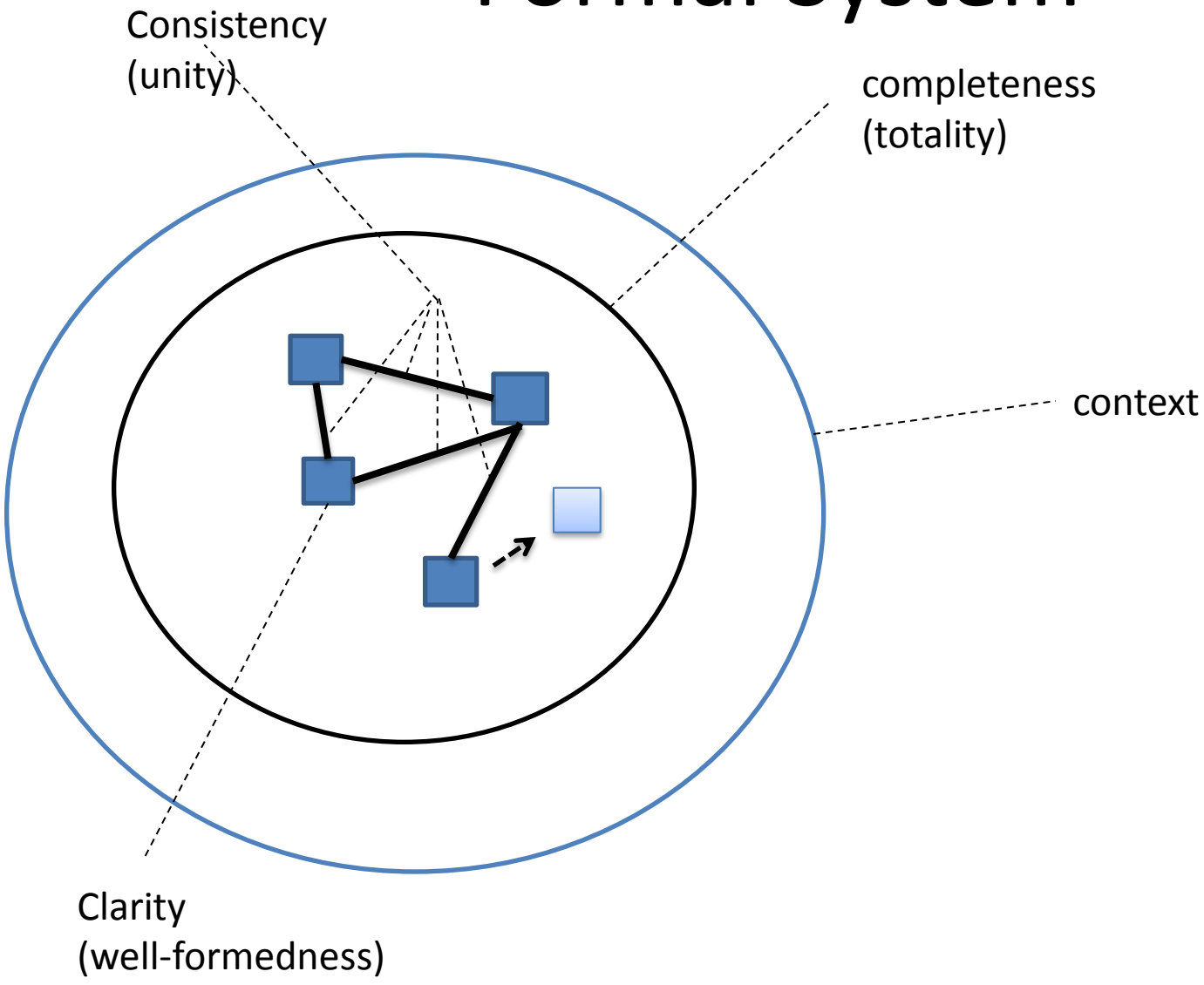
- Complex syntax for nesting
- Closed Turing Complete
- Too general
- Semantics determined by compiler actions

- Minimal Methods
- Graphical Representation
- Profiles for Extension
- Monolithic without unity
- Semantically Weak

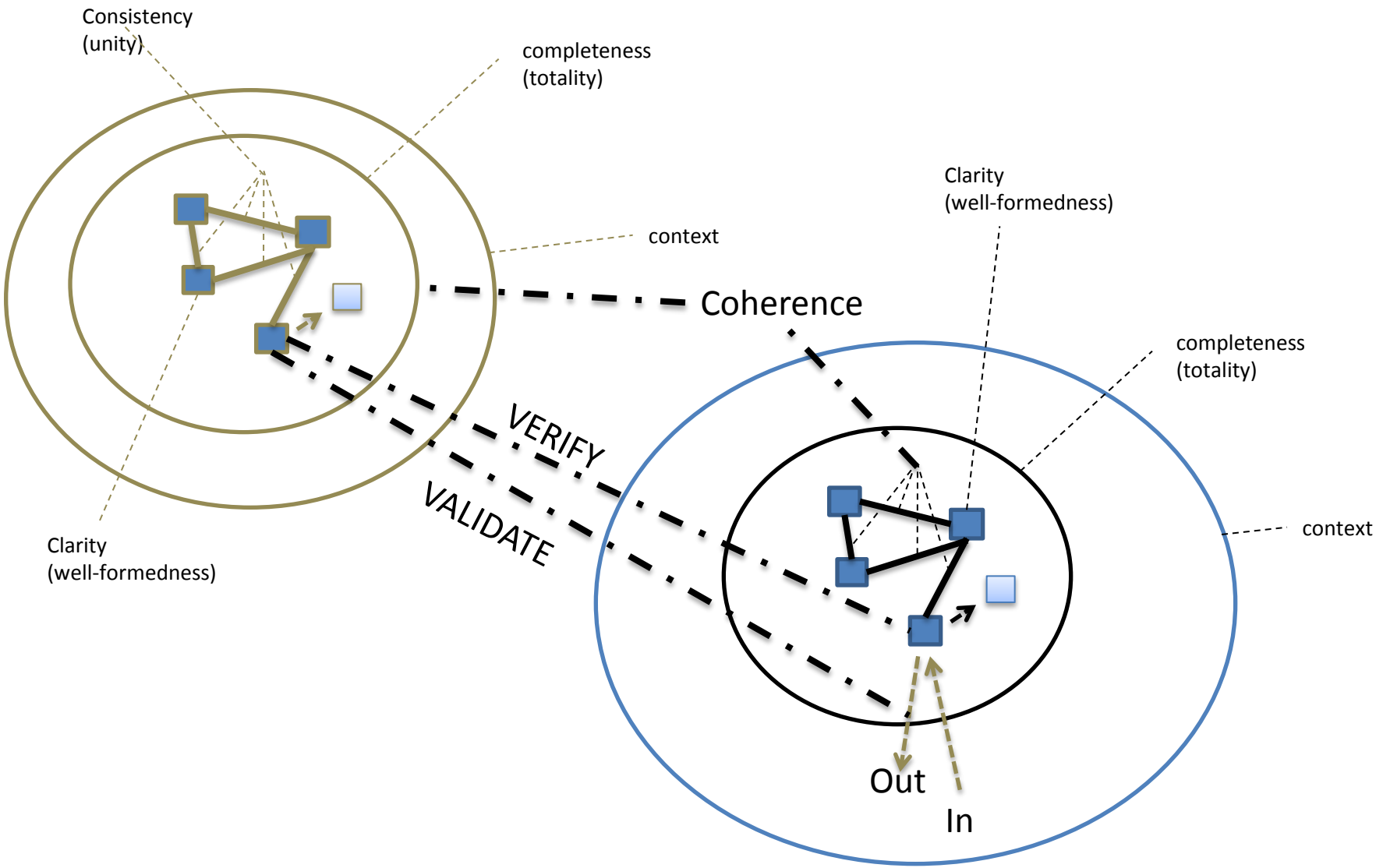
A Solution

- Wittgenstein's Tractatus
- Unbind design language from programming language for the design activity
- Simplify Language Structure without nesting
- Allow multiple connections in a single statement – more expressive and synthetic
- Semantics = Knowledge capture rather than complier execution
- Give up primary Visual representation mode
- Give up primary goal of parse-ablity

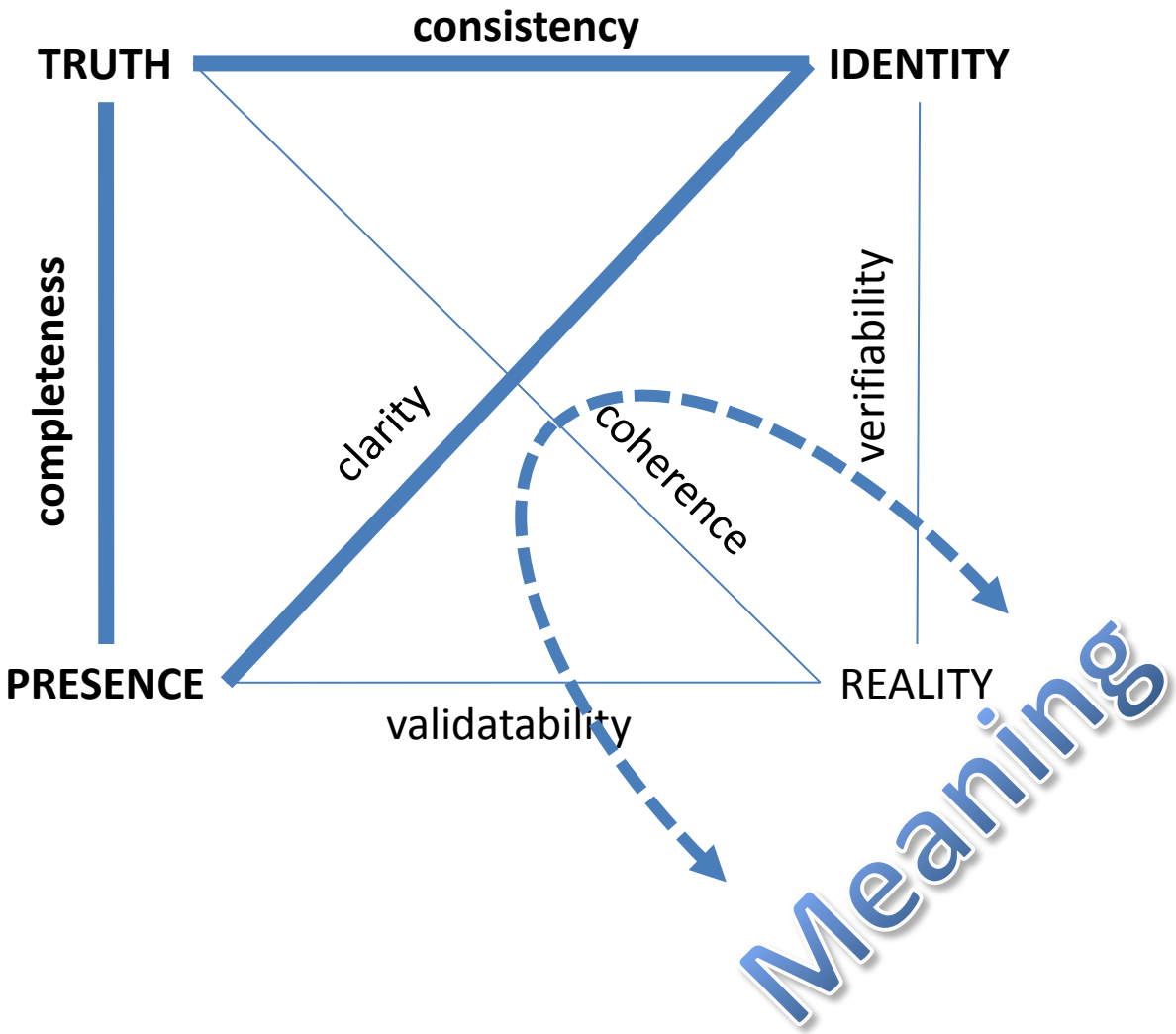
Formal System



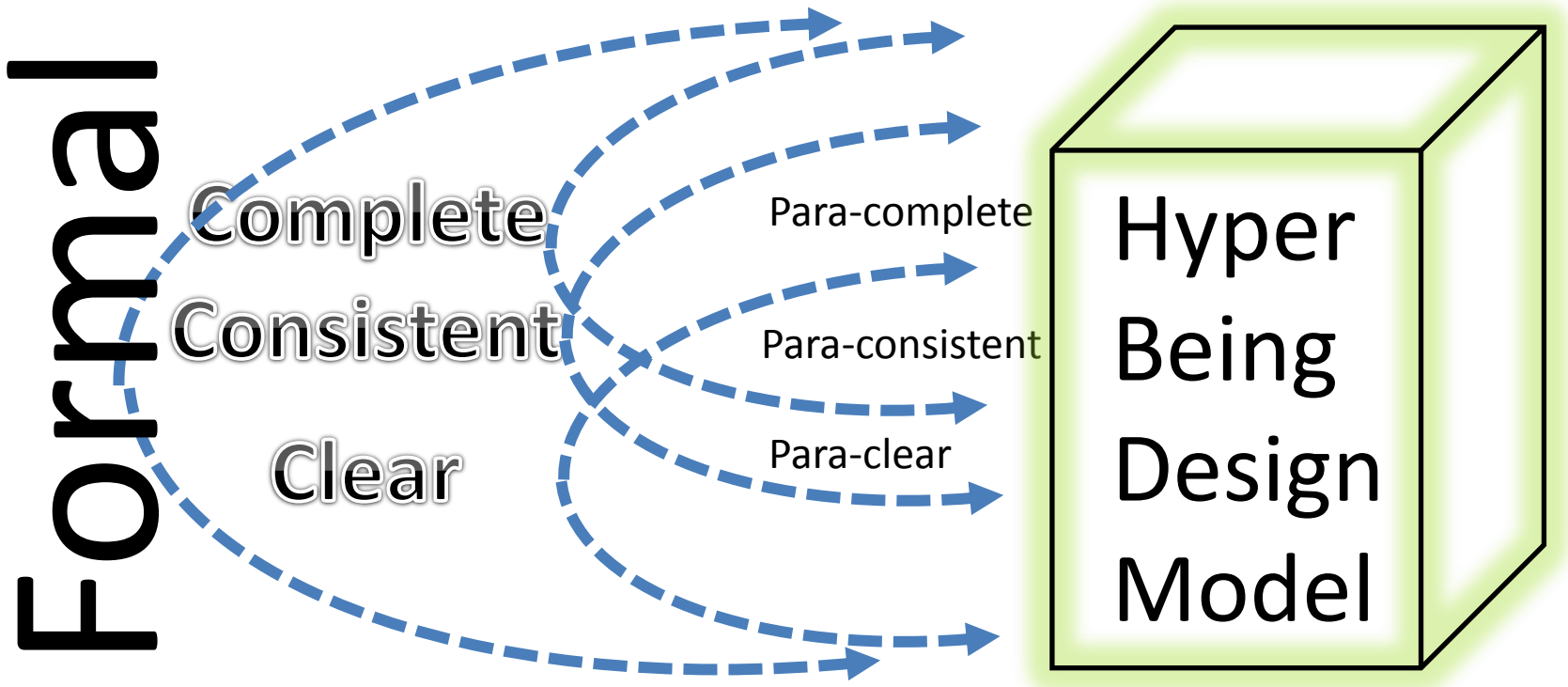
Add Reality to the Formal Model

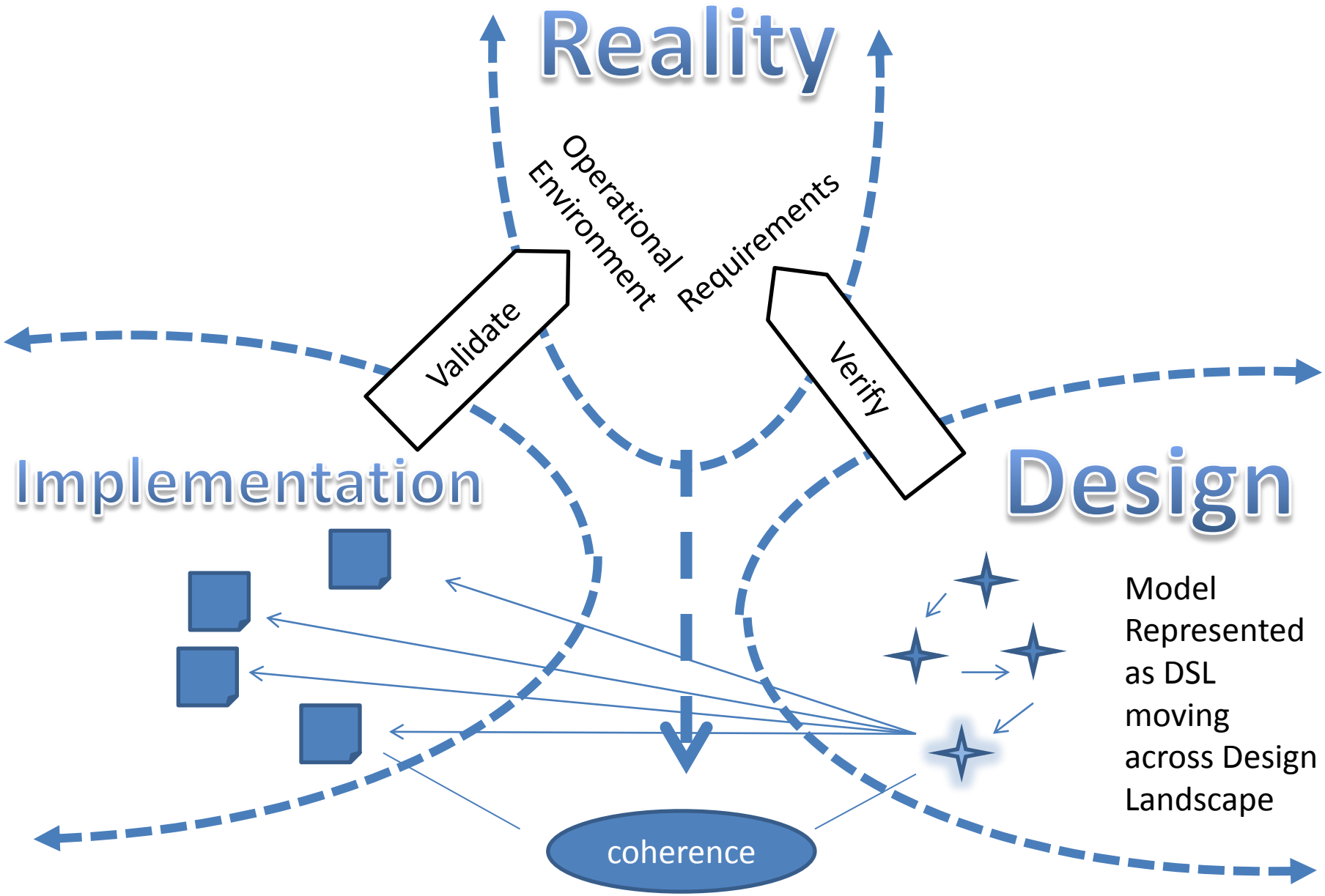


Aspects and Properties



Para-Formality




Reality
Operational Environment
Requirements
Validate
Verify
Implementation
Design
Model Represented as DSL moving across Design Landscape
coherence

People can *make up* languages more easily than they can learn already existing languages made up by others

- Fundamental Efficiency to be exploited in Domain Specific Language Model Development
- Schemas have the same structure as language and thus there is an internal coherence between language and schemas
- Language Oriented Approach unbinds Design Language from Programming language during design process
- Allows for Para-Properties related to all Aspects of Being
  - Para-
    - Consistency
    - Completeness
    - Clarity
    - Verification
    - Validation
    - Coherence

Math of Certainty	Kinds of Being
Determinate	Pure
Probability	Process
Fuzzy	Hyper
Propensity	Wild

Design of Design

- Multiple relationships for one statement
- Textual not graphical, at first
- Exploit Synthesis not just relations
- Precission vs. Precision
- Synthesis vs. Analysis
- Express each fact about design as it is known
  - Lacks complete knowledge to start with
- Use language template in Spreadsheet
  - Retooling not necessary – tool already available
  - Eases adoption of the core of Domain Specific Model Development

Snumber	Language Statement Number	CML01
Sname	Statement Name	EdgeIdentifer
arrow	decoration	->
Family	Language Family	CORE
Language	Domain Specific Language Name	Meta
Package	Package Name	language
id	identifier	*id*
Operator	Operator	Define
Switch	Switch	
Noun0/Operand0	Operand / Noun	identifier
id	identifier	*id*
Verb1	Verb	is
Preposition1/Adverb1	Preposition	
Noun1	Noun	edge
id	identifier	n
Qualifier1	Qualifier	
Verb2	Verb	
Preposition2	Preposition	
Noun2	Noun	
id	identifier	
Qualifier2	Qualifier	
Verb3	Verb	
Preposition3	Preposition	
Noun3	Noun	
id	identifier	
Qualifier4	Qualifier	
Verb4	Verb	
Preposition4	Preposition	
Noun4	Noun	
id	identifier	
Qualifier4	Qualifier	
period	.	.
znumber	Permanent Identifier	z1

Prefix

Operator/
Operand

SVpOq

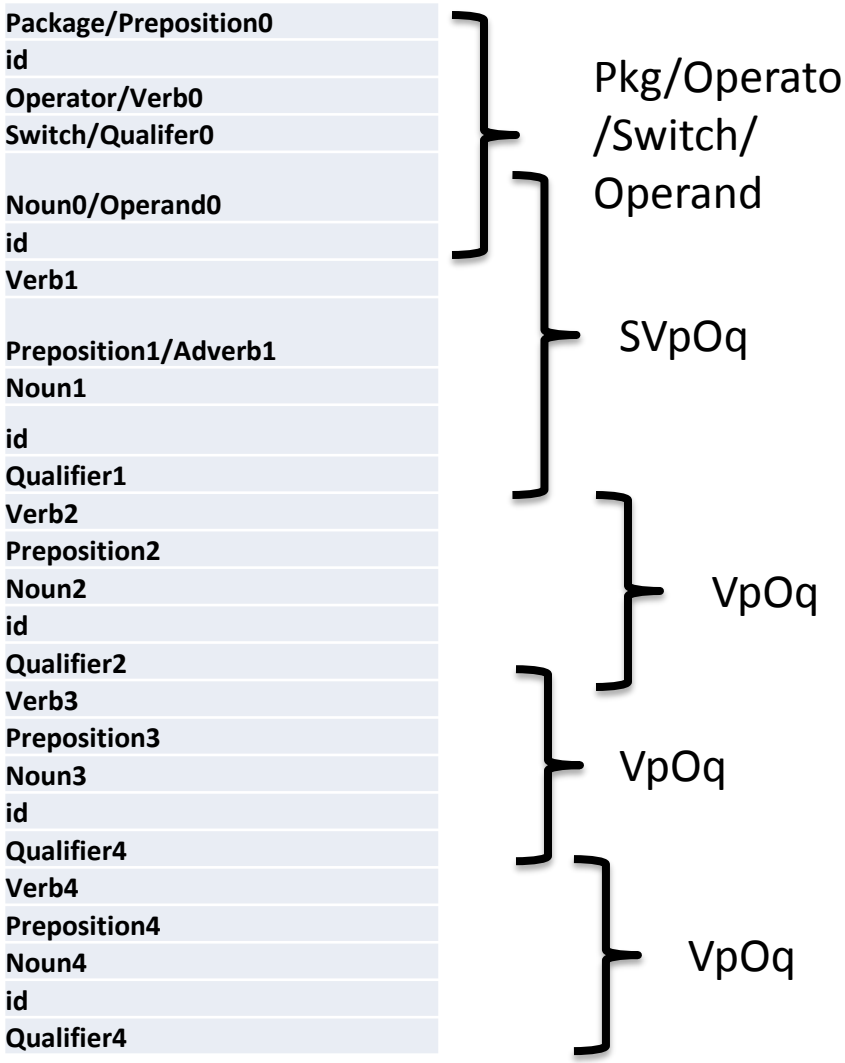
VpOq

VpOq

VpOq

Postfix

Four-way synthesis possible in each statement



Verb	Preposition	Noun	Qualifier
Op^0	Pkg^0 (in)	S^0/ Opnd	$Switch^0$
V^1	P^1	O^1	q^1
V^2	P^2	O^2	q^2
V^3	P^3	O^3	q^3
V^4	P^4	O^4	q^4

Meta[2]model = Spreadsheet language structure

Snumber	Sname	arrow	Family	Language	Package	id	Operator	Switch	Noun0/Operand0	id	Verb1	Preposition1/Adverb1	Noun1	id	Qualifier1	Verb2	Preposition2	Noun2	id	Qualifier2	Verb3	Preposition3	Noun3	id	Qualifier4	Verb4	Preposition4	Noun4	id	Qualifier4	period	znumber

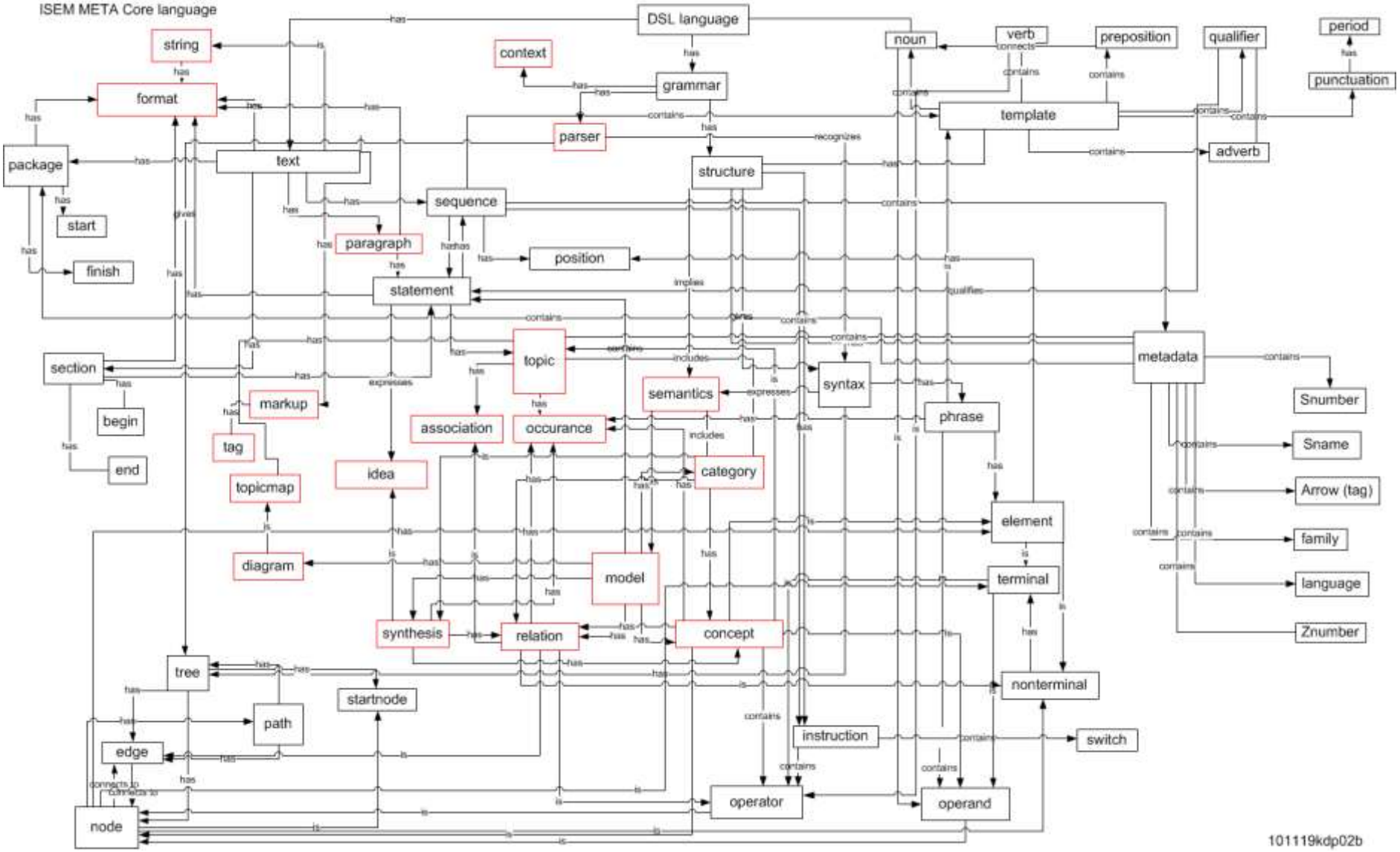
Meta[1]model = Domain Specific Language (DSL)

CML01	->	CORE	Meta	language	id	Define	identifier	id	is	edge	n																		.	z1	

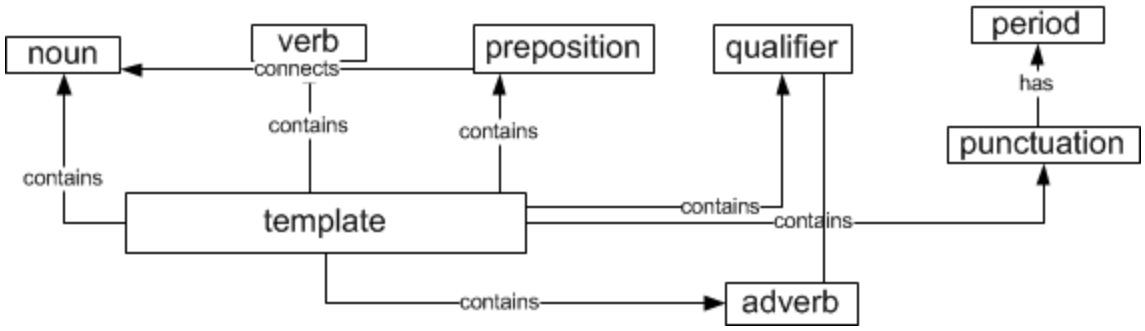
Meta[0]model = Instantiated Statements in DSL

CML01	->	CORE	Meta	language	*ISEM*	Define	identifier	*T1*	is	edge	*6*																		.	z1	

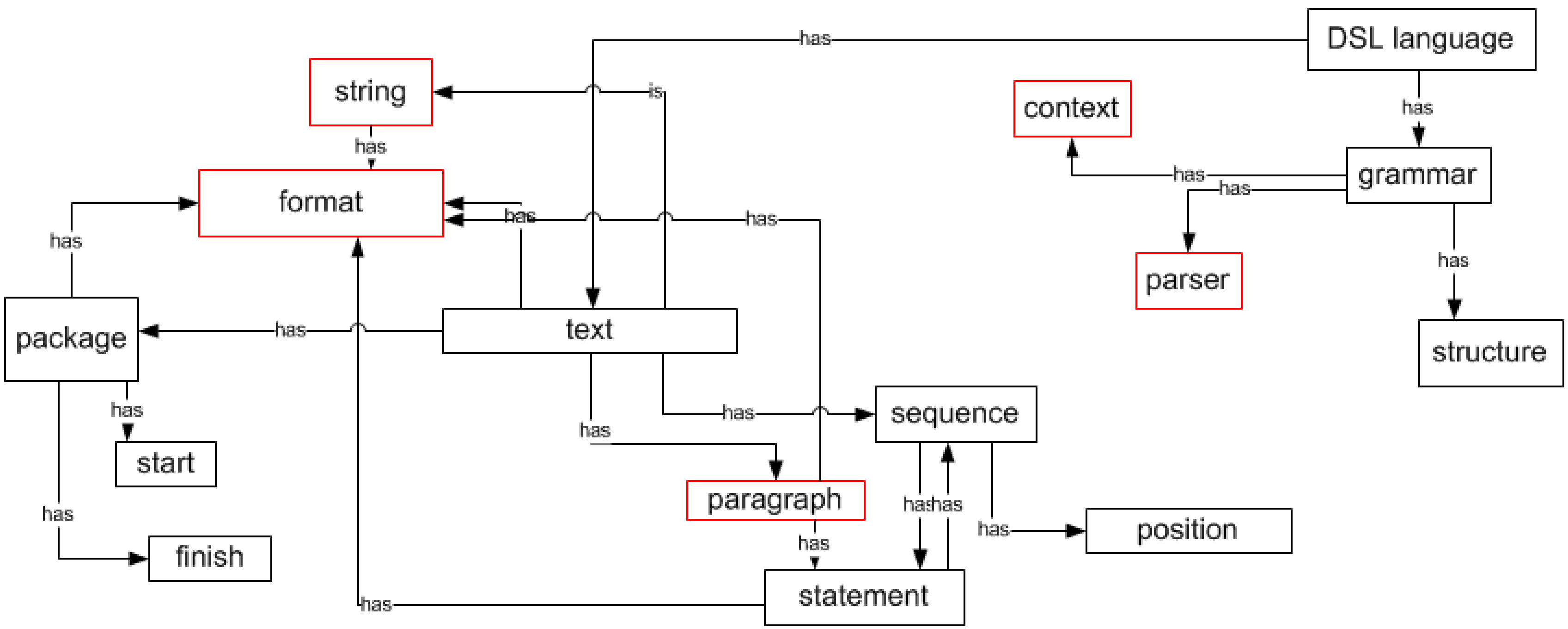
Reflexive Language about the ISEM Language



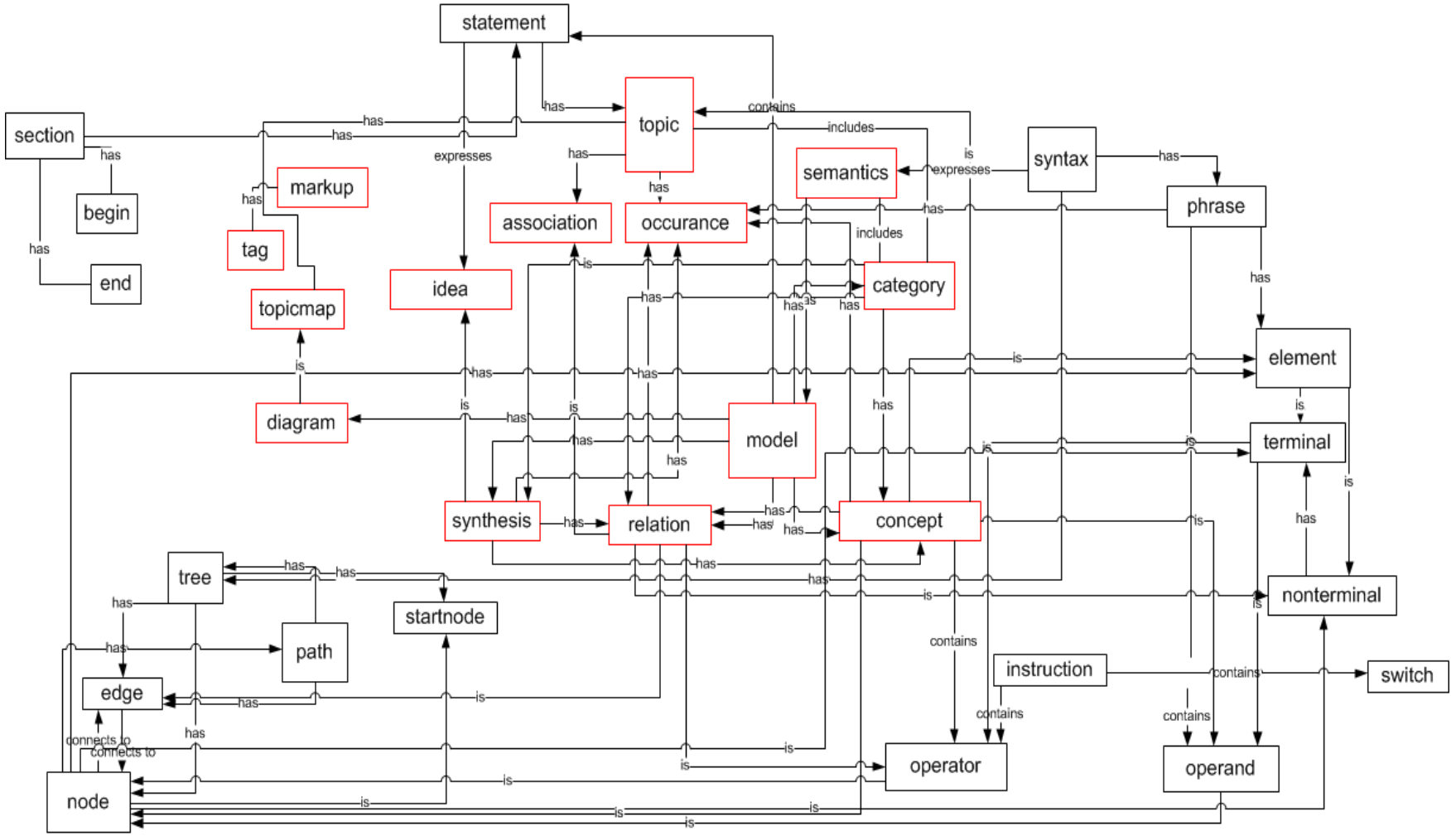
SVpOq Template



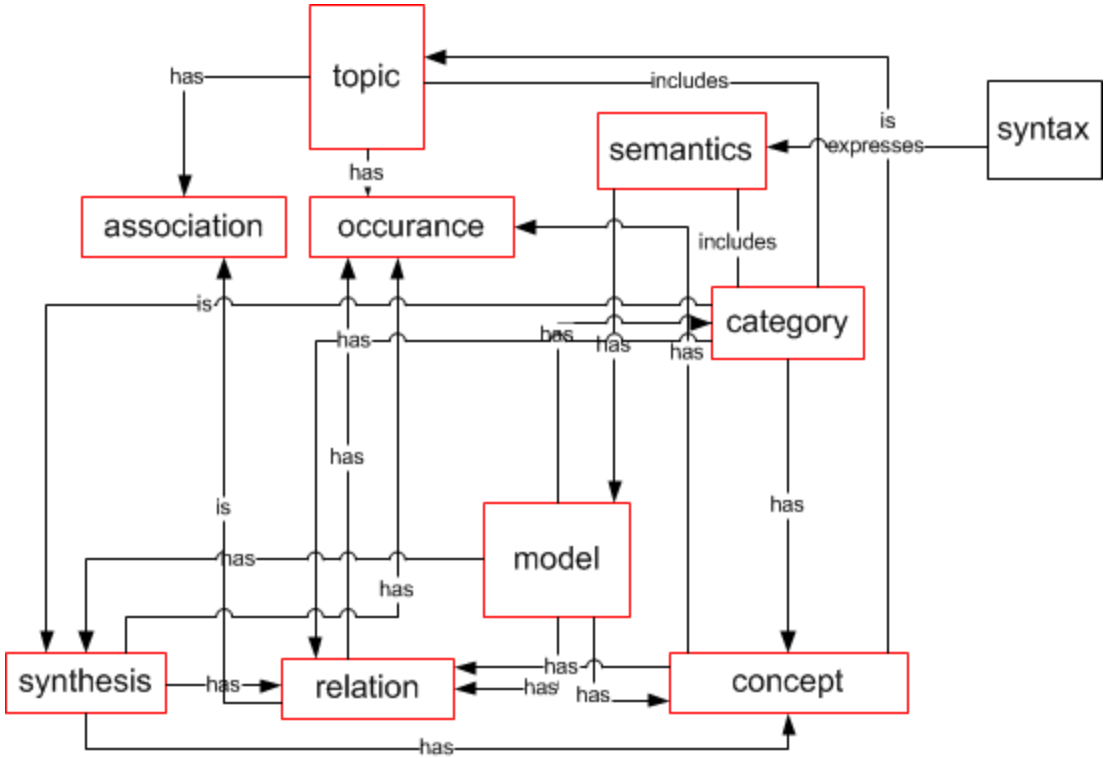
Language as structure and as text



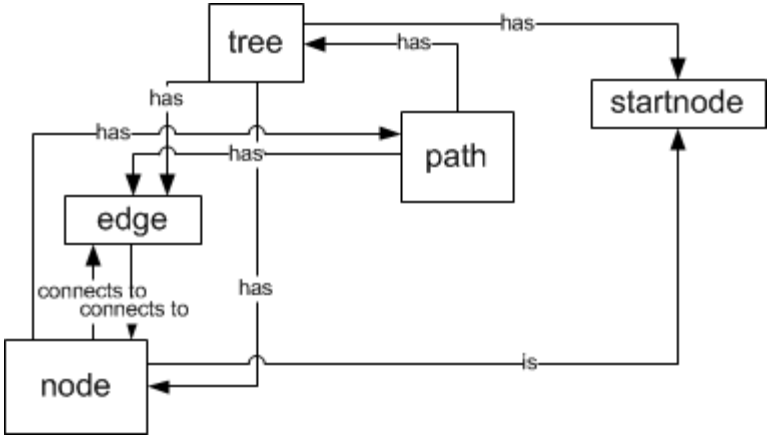
Statements of Design facts are the Basis of the Language



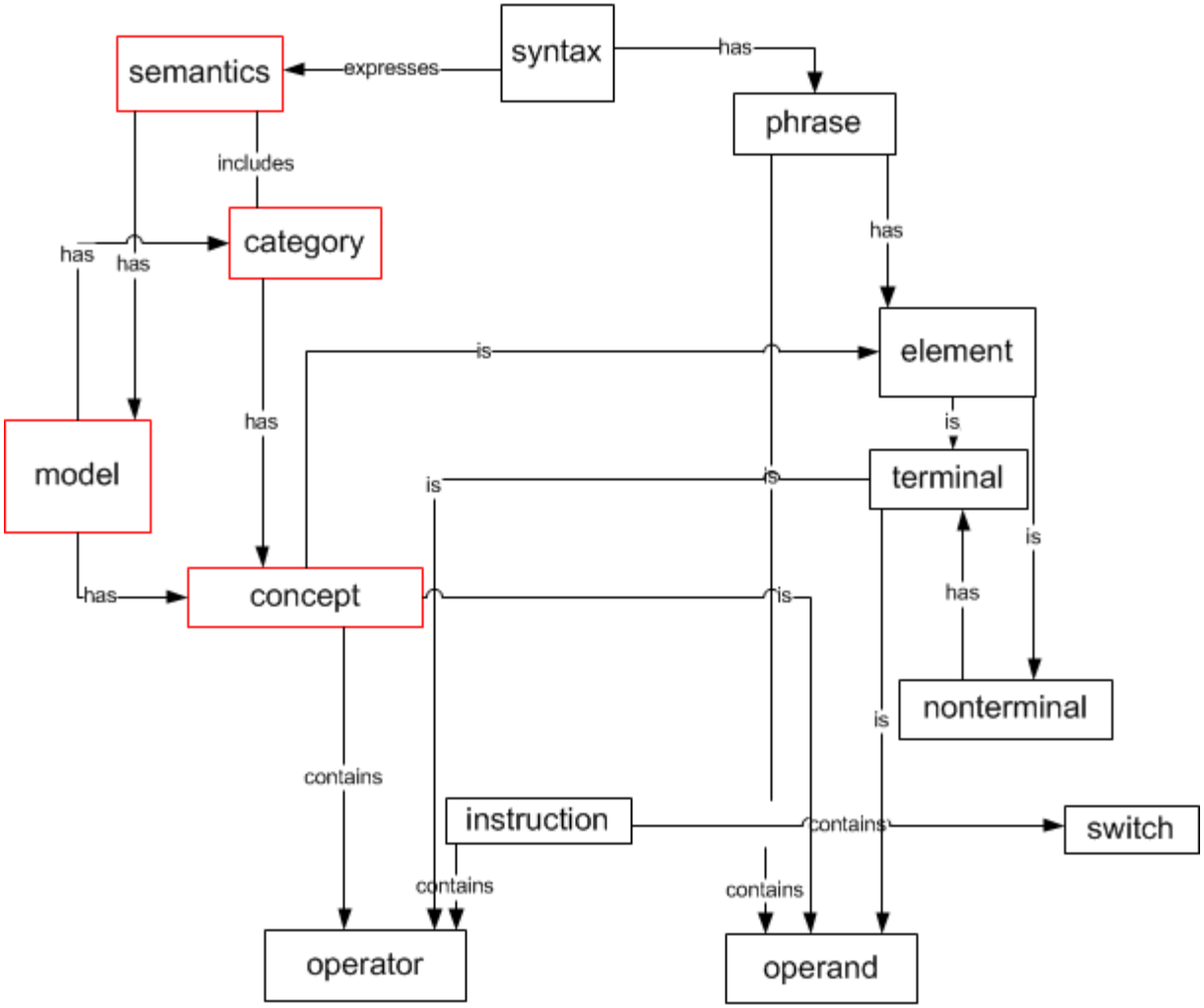
Topic Maps included in language description to give access to Models



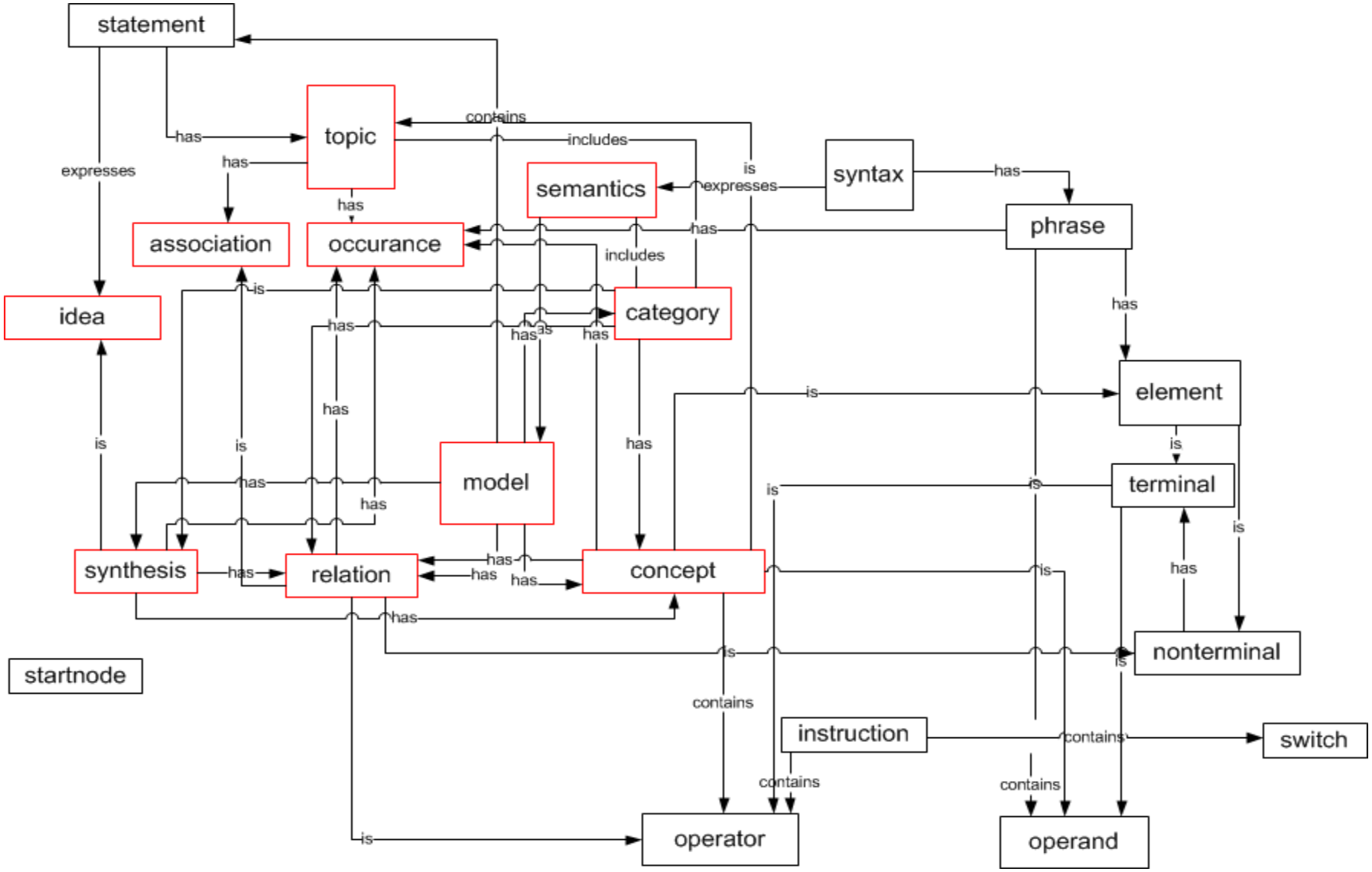
AST tree representation included



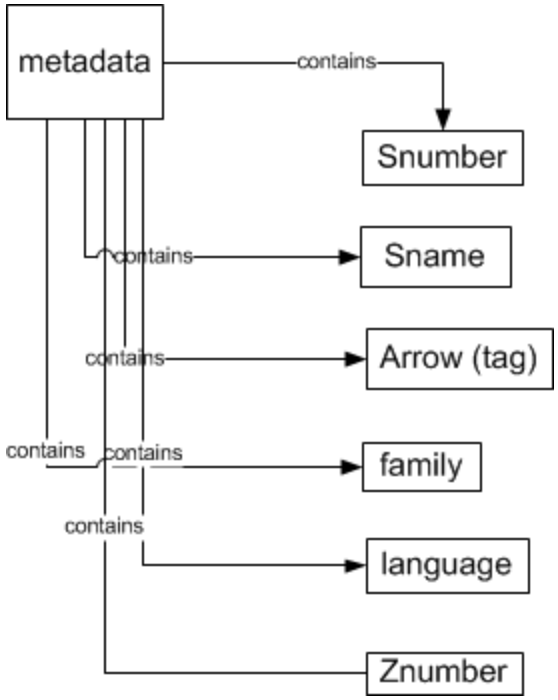
Operator and Operand structure



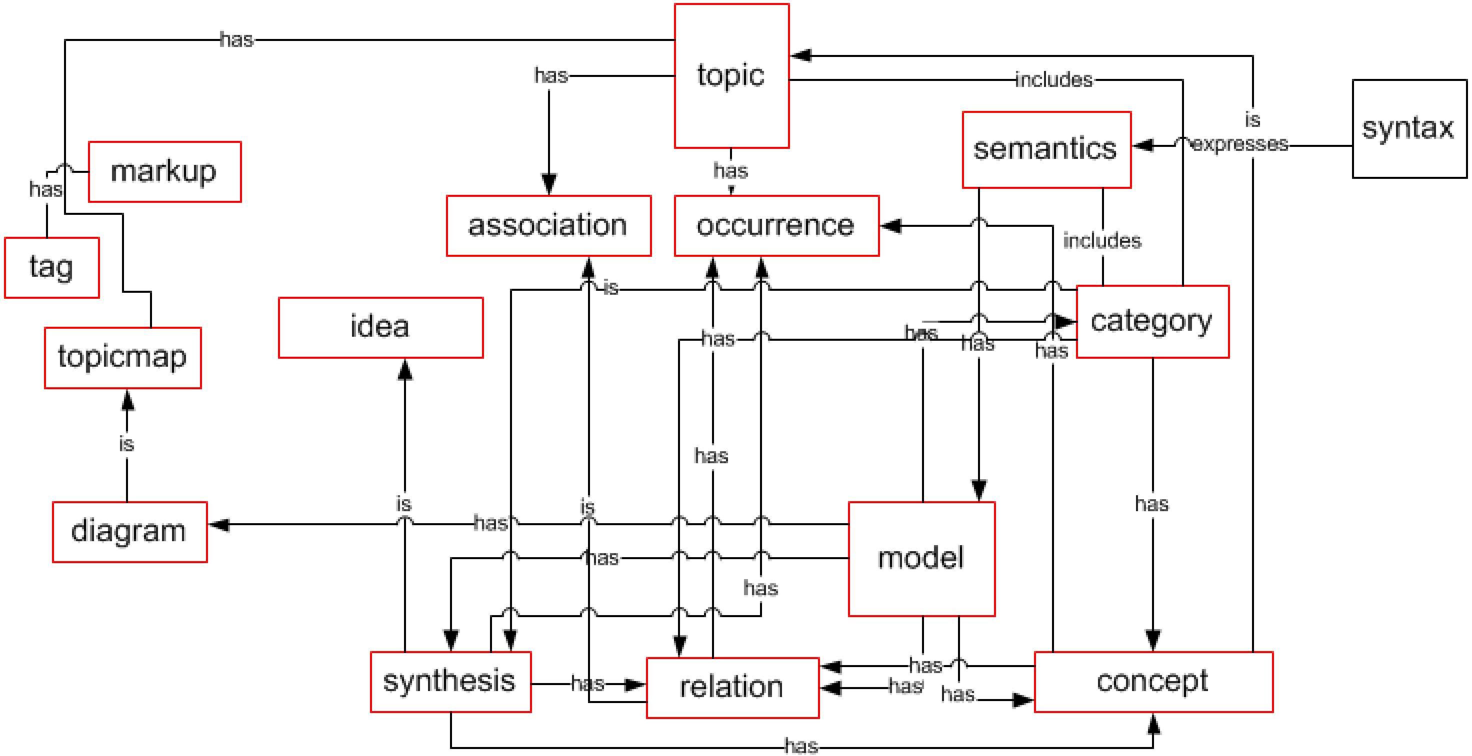
Semantics explicitly represented



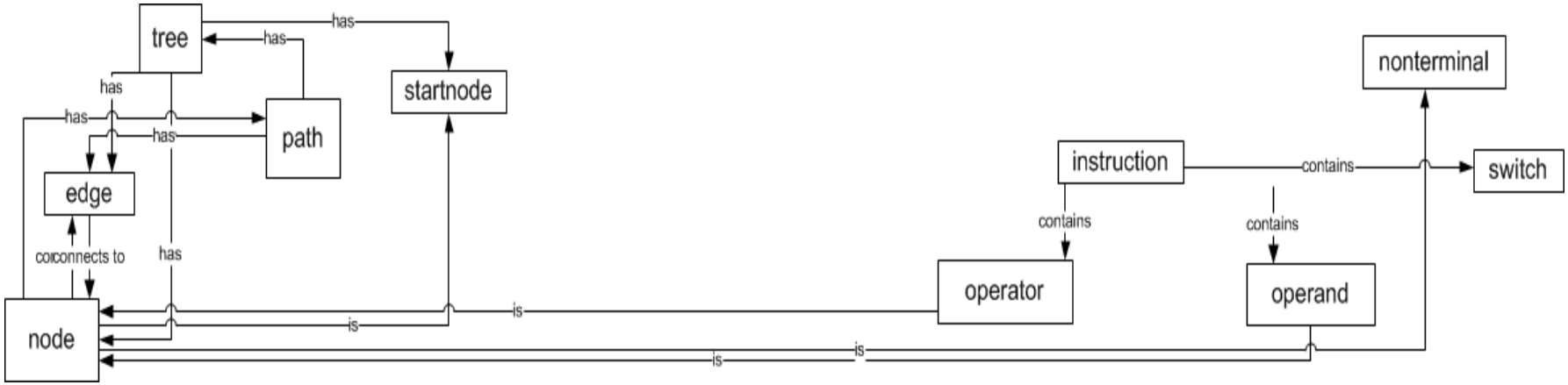
Metadata


metadata
contains
Snumber
contains
Sname
contains
Arrow (tag)
contains
contains
family
contains
language
Znumber

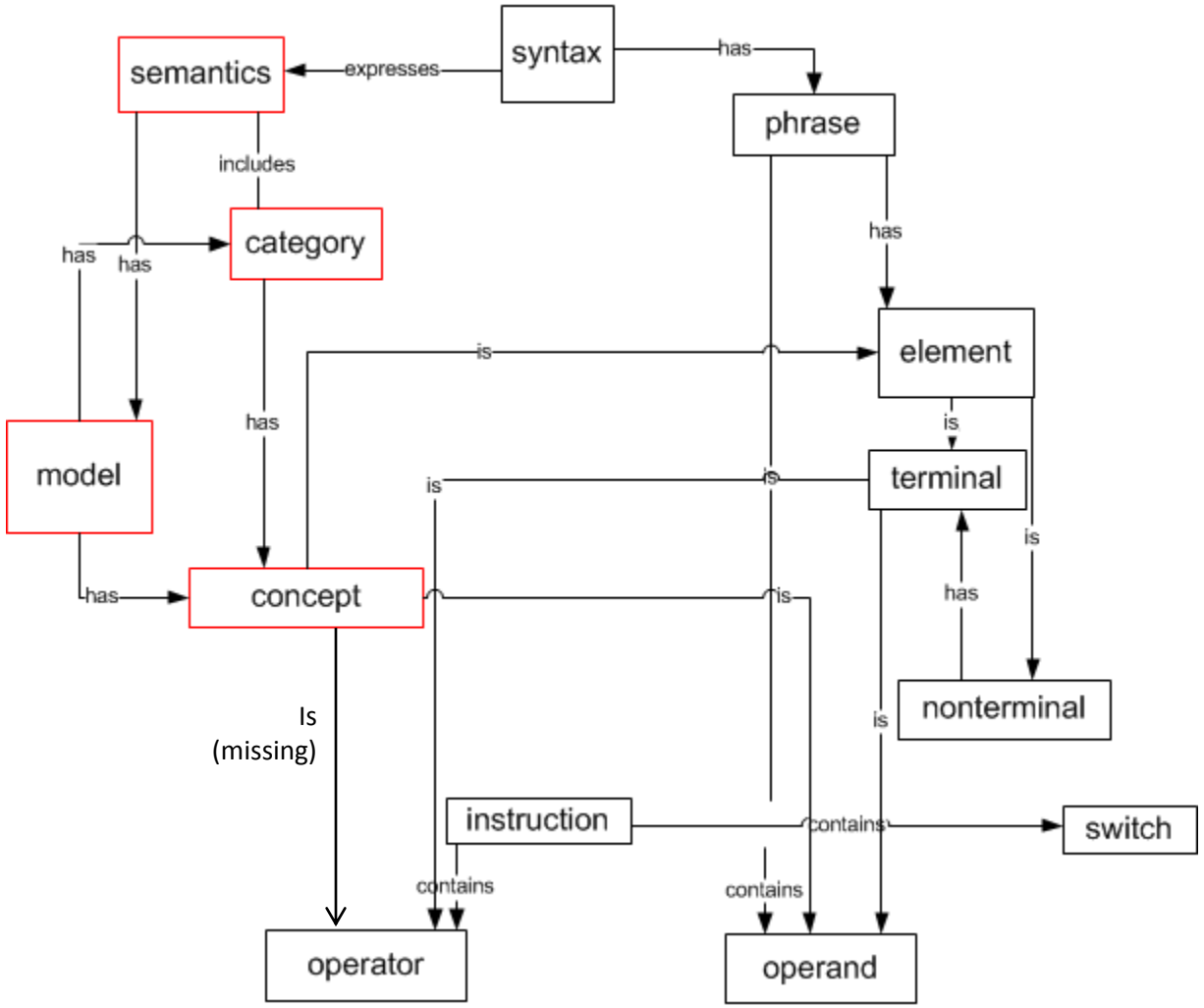
Recently added concepts which show relation between topics and model

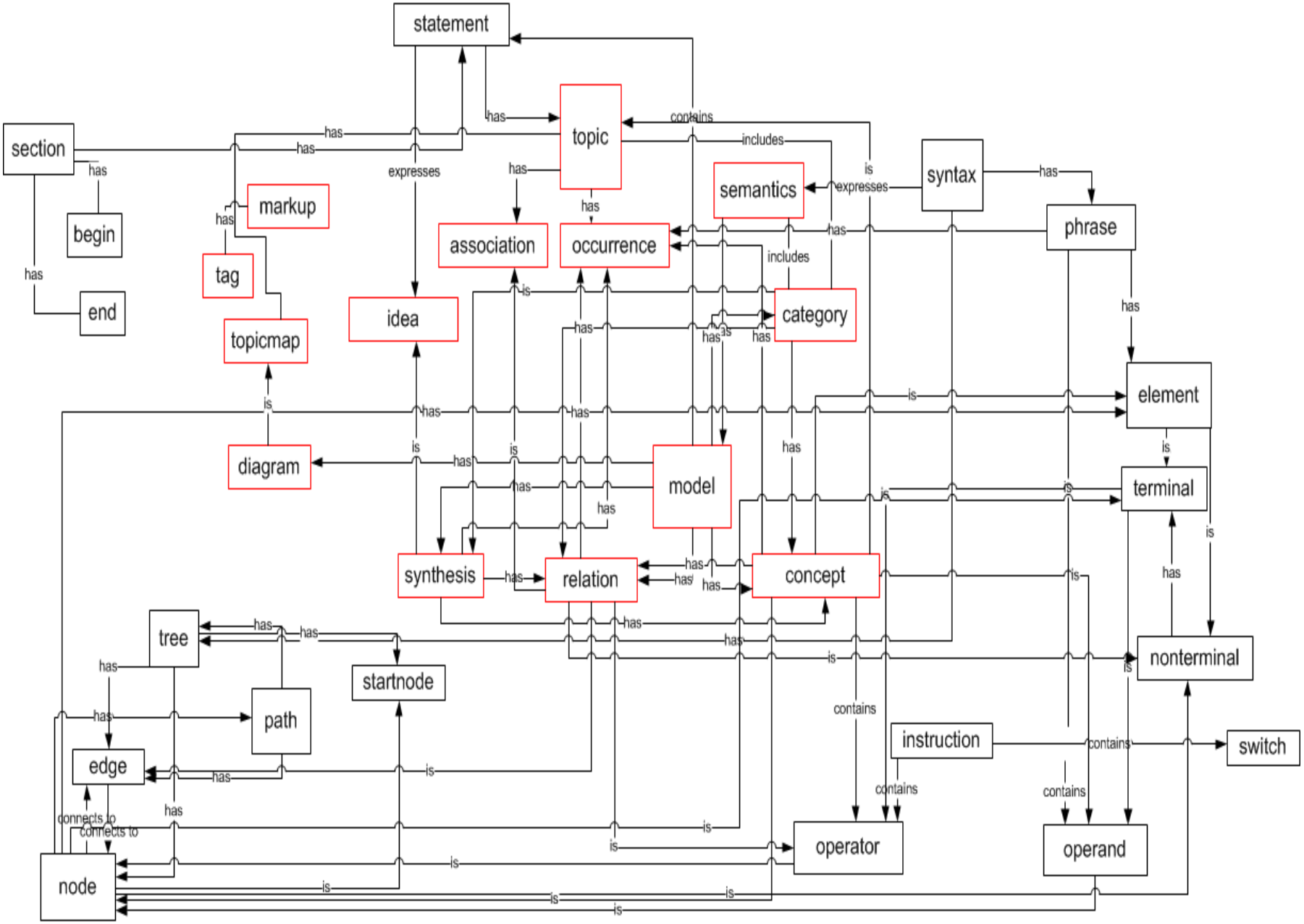


Relation between operators and operands and AST tree



Combination of Syntax and Semantics fully defines the language




statement
topic
section
begin
end
markup
tag
topicmap
idea
association
occurrence
semantics
syntax
phrase
category
element
terminal
nonterminal
diagram
model
synthesis
relation
concept
tree
startnode
path
edge
node
instruction
switch
operator
operand
has
is
expresses
includes
contains
connects to

Snumber	Sname	arrow	Family	Language	Package	id	Operator	Switch	Noun0/Operand0	id	Verb1	Preposition1/Adverb 1	Noun1	id	Qualifier1	Verb2
CML01		->	CORE	Meta	language	id	Define		identifier	id	is		**edge**	n		
CML02		->	CORE	Meta	language	id	Define		identifier	id	is		**element**		string	
CML03		->	CORE	Meta	language	id	Define		identifier	id	is		**grammar**			
CML04		->	CORE	Meta	language	id	Define		identifier	id	is		**instruction**			
CML05		->	CORE	Meta	language		Define		identifier	id	is		**language**			
CML06		->	CORE	Meta	language	id	Define		identifier	id	is		**metadata**			
CML07		->	CORE	Meta	language	id	Define		identifier	id	is		**node**	n		
CML08		->	CORE	Meta	language	id	Define		identifier	id	is		**nonterminal**	n		
CML09		->	CORE	Meta	language	id	Define		identifier	id	is		**noun**			
CML10		->	CORE	Meta	language	id	Define		identifier	id	is		**operand**			
CML11		->	CORE	Meta	language	id	Define		identifier	id	is		**operator**			
CML12		->	CORE	Meta	language	id	Define		identifier	id	is		**package**			
CML13		->	CORE	Meta	language	id	Define		identifier	id	is		**path**		list	
CML14		->	CORE	Meta	language	id	Define		identifier	id	is		**phrase**			
CML15		->	CORE	Meta	language	id	Define		identifier	id	is		**position**	n		
CML16		->	CORE	Meta	language	id	Define		identifier	id	is		**punctuation**			
CML17		->	CORE	Meta	language	id	Define		identifier	id	is		**qualifier**		string	
CML18		->	CORE	Meta	language	id	Define		identifier	id	is		**section**	n		
CML19		->	CORE	Meta	language	id	Define		identifier	id	is		**sequence**		list	
CML20		->	CORE	Meta	language	id	Define		identifier	id	is		**statement**		string	
CML21		->	CORE	Meta	language	id	Define		identifier	id	is		**structure**			
CML22		->	CORE	Meta	language	id	Define		identifier	id	is		**switch**		string	
CML23		->	CORE	Meta	language	id	Define		identifier	id	is		**syntax**			
CML24		->	CORE	Meta	language	id	Define		identifier	id	is		**template**			
CML25		->	CORE	Meta	language	id	Define		identifier	id	is		**terminal**	n		
CML26		->	CORE	Meta	language	id	Define		identifier	id	is		**text**		string	
CML27		->	CORE	Meta	language	id	Define		identifier	id	is		**transform**		string	
CML28		->	CORE	Meta	language	id	Define		identifier	id	is		**tree**		hierarchy	
CML29		->	CORE	Meta	language	id	Define		identifier	id	is		**verb**			

CML30	**token**		**is**		**adverb**			
CML31	**token**		**is**		**arrow**			
CML32	**token**		**is**		**begin**			
CML33	**token**		**is**		**end**			
CML34	**token**		**is**		**family**			
CML35	**token**		**is**		**finish**			
CML115	**token**		**is**		**paragraph**			
CML36	**token**		**is**		**period**		**"."**	
CML37	**token**		**is**		**preposition**			
CML38	**token**		**is**		**Sname**		**string**	
CML39	**token**		**is**		**Snumber**		**"aaannn"**	
CML40	**token**		**is**		**start**			
CML41	**token**		**is**		**Znumber**		**"Znnnn"**	

CML161	association	id	has		role	id		
CML170	category	id	has		concept	id		
CML171	category	id	has		relation	id		
CML173	category	id	includes		topic	id		
CML172	category	id	is		synthesis	id		
CML154	concept	id	has		concept	id		
CML166	concept	id	has		occurrence	id		
CML155	concept	id	has		relation	id		
CML148	concept	id	is		element	id		
CML147	concept	id	is		node	id		
CML150	concept	id	is		operand	id		
CML164	concept	id	is		topic	id		
CML43	edge	id	connects	to	node	id		
CML44	element	id	is		nonterminal	id		
CML45	element	id	is		terminal	id		
CML46	element	id	has		position	n		
CML122	grammar	id	has		context	id		
CML125	grammar	id	has		parser	id		
CML47	grammar	id	has		structure	id		
CML48	instruction	id	contains		operand	id		
CML49	instruction	id	contains		operator	id		
CML50	instruction	id	contains		switch	id		
CML51	language	id	has		text	id		
CML52	language	id	has		grammar	id		
CML144	markup	id	has		tag	id		
CML53	metadata	id	contains		arrow	id		

CML54	->	CORE	Meta	language	id	Posit	metadata	id	contains	family	id	for	statement	string	.
CML55	->	CORE	Meta	language	id	Posit	metadata	id	contains	language	id	for	statement	string	.
CML56	->	CORE	Meta	language	id	Posit	metadata	id	contains	package	id	for	statement	string	.
CML57	->	CORE	Meta	language	id	Posit	metadata	id	contains	Sname	id	of	statement	string	.
CML58	->	CORE	Meta	language	id	Posit	metadata	id	contains	Snumber	id	of	statement	string	.
CML177	->	CORE	Meta	language	id	Posit	metadata	id	contains	topic	id				.
CML59	->	CORE	Meta	language	id	Posit	metadata	id	contains	Znumber	id	of	statement	string	.

CML140		->	CORE	Meta	language	id	Posit		model	id	contains		statement	id
CML169		->	CORE	Meta	language	id	Posit		model	id	has		category	id
CML129		->	CORE	Meta	language	id	Posit		model	id	has		concept	id
CML179		->	CORE	Meta	language	id	Posit		model	id	has		diagram	id
CML130		->	CORE	Meta	language	id	Posit		model	id	has		relation	id
CML131		->	CORE	Meta	language	id	Posit		model	id	has		synthesis	id
CML60		->	CORE	Meta	language	id	Posit		node	id	connects	to	edge	id
CML61		->	CORE	Meta	language	id	Posit		node	id	has		element	id
CML62		->	CORE	Meta	language	id	Posit		node	id	is	start	node	id
CML63		->	CORE	Meta	language	id	Posit		node	id	is		nonterminal	id
CML64		->	CORE	Meta	language	id	Posit		node	id	is		operand	id
CML65		->	CORE	Meta	language	id	Posit		node	id	is		operator	id
CML66		->	CORE	Meta	language	id	Posit		node	id	is		terminal	id
CML67		->	CORE	Meta	language	id	Posit		nonterminal	id	has		nonterminal	id
CML68		->	CORE	Meta	language	id	Posit		nonterminal	id	has		terminal	id
CML69		->	CORE	Meta	language	id	Posit		noun	id	is		operand	id
CML70		->	CORE	Meta	language	id	Posit		operand	id	is		terminal	id
CML71		->	CORE	Meta	language	id	Posit		operator	id	is		terminal	id
CML72		->	CORE	Meta	language	id	Posit		package	id	has		finish	id
CML141		->	CORE	Meta	language	id	Posit		package	id	has		format	id
CML73		->	CORE	Meta	language	id	Posit		package	id	has		start	id

CML73		->	CORE	Meta	language	id	Posit		package	id	has		start	id
CML74		->	CORE	Meta	language	id	Posit		package	id			finish	
CML75		->	CORE	Meta	language	id	Posit		package	id			start	
CML135		->	CORE	Meta	language	id	Posit		paragraph	id	has		format	id
CML134		->	CORE	Meta	language	id	Posit		paragraph	id	has		separator	id
CML118		->	CORE	Meta	language	id	Posit		paragraph	id	has		statement	id
CML120		->	CORE	Meta	language	id	Posit		paragraph	id			completion	
CML119		->	CORE	Meta	language	id	Posit		paragraph	id			inception	
CML126		->	CORE	Meta	language	id	Posit		parser	id	gives		tree	id
CML145		->	CORE	Meta	language	id	Posit		parser	id	recognizes		syntax	id
CML76		->	CORE	Meta	language	id	Posit		path	id	has		edge	id
CML77		->	CORE	Meta	language	id	Posit		path	id	has		node	id
CML78		->	CORE	Meta	language	id	Posit		phrase	id	has		element	id
CML176		->	CORE	Meta	language	id	Posit		phrase	id	has		occurrence	id
CML79		->	CORE	Meta	language	id	Posit		phrase	id	is		instruction	id
CML80		->	CORE	Meta	language	id	Posit		phrase	id	is		template	id
CML81		->	CORE	Meta	language	id	Posit		preposition	id	connects		noun	id
CML82		->	CORE	Meta	language	id	Posit		punctuation	id	is		period	id
CML83		->	CORE	Meta	language	id	Posit		qualifier	id	qualifies		statement	id
CML163		->	CORE	Meta	language	id	Posit		relation	id	is		association	id
CML146		->	CORE	Meta	language	id	Posit		relation	id	is		edge	id
CML149		->	CORE	Meta	language	id	Posit		relation	id	is		operator	id
CML168		->	CORE	Meta	language	id	Posit		relation	id	has		occurrence	id
CML151		->	CORE	Meta	language	id	Posit		relation	id	is		nonterminal	id
CML84		->	CORE	Meta	language	id	Posit		section	id	has		begin	id
CML85		->	CORE	Meta	language	id	Posit		section	id	has		end	id
CML137		->	CORE	Meta	language	id	Posit		section	id	has		format	id
CML86		->	CORE	Meta	language	id	Posit		section	id	has		statement	id

CML87		->	CORE	Meta	language	id	Posit		section	id			begin						
CML88		->	CORE	Meta	language	id	Posit		section	id			end						
CML124		->	CORE	Meta	language	id	Posit		semantics	id	has		model	id					
CML175		->	CORE	Meta	language	id	Posit		semantics	id	includes		category	id					
CML89		->	CORE	Meta	language	id	Posit		sequence	id	contains		instruction	id					
CML90		->	CORE	Meta	language	id	Posit		sequence	id	contains		metadata	id					
CML91		->	CORE	Meta	language	id	Posit		sequence	id	contains		template	id					
CML165		->	CORE	Meta	language	id	Posit		sequence	id	has		statement	id					
CML92		->	CORE	Meta	language	id	Posit		sequence	id	has		position	n					
CML138		->	CORE	Meta	language	id	Posit		statement	id	expresses		idea	id					
CML133		->	CORE	Meta	language	id	Posit		statement	id	gives		idea	id					
CML93		->	CORE	Meta	language	id	Posit		statement	id	has		sequence	id					
CML156		->	CORE	Meta	language	id	Posit		statement	id	has		topic	id					
CML142		->	CORE	Meta	language	id	Posit		string	id	has		format	id					
CML94		->	CORE	Meta	language	id	Posit		structure	id	gives		syntax	id			with	transform	id

CML95		->	CORE	Meta	language	id	Posit		structure	id	has		instruction	id
CML96		->	CORE	Meta	language	id	Posit		structure	id	has		metadata	id
CML123		->	CORE	Meta	language	id	Posit		structure	id	has		semantics	id
CML97		->	CORE	Meta	language	id	Posit		structure	id	has		template	id
CML174		->	CORE	Meta	language	id	Posit		syntax	id	expresses		semantics	id
CML98		->	CORE	Meta	language	id	Posit		syntax	id	has		phrase	id
CML99		->	CORE	Meta	language	id	Posit		syntax	id	has		tree	id
CML153		->	CORE	Meta	language	id	Posit		synthesis	id	has		concept	id
CML167		->	CORE	Meta	language	id	Posit		synthesis	id	has		occurrence	id
CML152		->	CORE	Meta	language	id	Posit		synthesis	id	has		relation	id
CML132		->	CORE	Meta	language	id	Posit		synthesis	id	is		idea	id
CML100		->	CORE	Meta	language	id	Posit		template	id	contains		adverb	id
CML101		->	CORE	Meta	language	id	Posit		template	id	contains		noun	id
CML102		->	CORE	Meta	language	id	Posit		template	id	contains		preposition	id
CML103		->	CORE	Meta	language	id	Posit		template	id	contains		punctuation	id
CML104		->	CORE	Meta	language	id	Posit		template	id	contains		qualifier	id
CML105		->	CORE	Meta	language	id	Posit		template	id	contains		verb	id
CML136		->	CORE	Meta	language	id	Posit		text	id	has		format	id
CML143		->	CORE	Meta	language	id	Posit		text	id	has		markup	id
CML106		->	CORE	Meta	language	id	Posit		text	id	has		package	id
CML107		->	CORE	Meta	language	id	Posit		text	id	has		sequence	id

CML121		->	CORE	Meta	language	id	Posit		text	id	has		paragraph	id
CML108		->	CORE	Meta	language	id	Posit		text	id	has		section	id
CML139		->	CORE	Meta	language	id	Posit		text	id	is		string	id
CML158		->	CORE	Meta	language	id	Posit		topic	id	has		association	id
CML157		->	CORE	Meta	language	id	Posit		topic	id	has		occurrence	id
CML160		->	CORE	Meta	language	id	Posit		topic	id	has		topicname	id
CML159		->	CORE	Meta	language	id	Posit		topicmap	id	contains		topic	id
CML178		->	CORE	Meta	language	id	Posit		topicmap	id	is		diagram	id
CML180		->	CORE	Meta	language	id	Posit		topicmap	id	is		diagram	id
CML162		->	CORE	Meta	language	id	Posit		topicname	id	has		variant	id
CML109		->	CORE	Meta	language	id	Posit		tree	id	has		edge	id
CML110		->	CORE	Meta	language	id	Posit		tree	id	has		node	id
CML111		->	CORE	Meta	language	id	Posit		tree	id	has	start	node	id
CML112		->	CORE	Meta	language	id	Posit		tree	id	has		path	id
CML127		->	CORE	Meta	language	id	Posit		tree	id	is		abstract	
CML128		->	CORE	Meta	language	id	Posit		tree	id	is		concrete	
CML113		->	CORE	Meta	language	id	Posit		verb	id	is		operator	id
CML114		->	CORE	Meta	language	id	start							

Modus Operendi

- Make up language using existing languages as examples
- Existing languages cover architectural subjects
- Domain languages should be patterned on architectural languages
- Allow single facts to be recorded
- Give consolidation statements at various levels as necessary
- Exercise for the Student
  - Take a complex design view chart and convert it into a language representation that captures the knowledge

Ontology and Domain Specific Languages

- By creating a language that describes an expert's view of a domain, we are producing an ontology of that domain by identifying significant nouns, verbs, qualifiers, and relationships in a synthesis
- The synthesis is the multiple relationships that can be obtained at the same time when brought together by the statements of the language

Features and DSLs

- Feature differences can be specified by configuration statements added to the language
- Feature trees allow for Product Line Engineering and Reuse between products
- A feature language could be created to be explicit in relation to the different features that various versions of the product embody

Aspects and DSLs

- Cross cutting concerns are easily implemented by small domain specific languages that mention orthogonal concerns
- Aspects are the duals of objects
- Mass vs. Set orientation

Models and DSLs

- A coherent model of a domain and its design architecture provides a complete synthesis of descriptive statements
- Models in Mathematics are all the true statements about a category.
- Thus the DSLs implement the model explicitly in the statements that are produced about a given object within the design
- Mathematical models must precede the implementation of programmatic models if the models are to be coherent

Parsers and DSLs

- Parsers can be written to read the design or domain language and produce database or network representations once the language has matured so that it is no longer radically changing
- Statement relationships can be checked and OCL constraints applied to the model after the flux of design has settled down

TextUML

- TextUML can be used to align with UML 2.0 and should be extended to align with SysML
- Using textual UML means we do not have to reinvent the UML2 in textual form
- Xtext textual UML converter can be used to produce graphical representations
- Human Readable UML is also available

Other considerations

- Textual representations can be CMed in a normal GIT or other repository
- OCL can be used to enforce constraints on the model
- Gurevich ASM and Wisse Metapattern methods can be used across all the schemas as a basis for modeling pre-design causality and computablity
- This offers a bridge from Requirements to Design
- Design occurs when performance questions begin to impact the representation
- Architectural Designs are the “staticware” that give a framework for detailed design

Other options

- DSLs can be developed into External Parsed DSLs
- They can be ported into DSL friendly languages such as Pi, Converge, M (Oslo), MPS.
- Other programing languages of interest (as infrastructure) are Mira, Scala, Fortress, Falcon, Factor (Forth)

At the Systems Engineering Level

- Use DSLs to capture knowledge.
- Emphasize cognitive coherence, not compile-ability
- Use GASM to show causality and computability
- Use Metapattern to derive objects from context
- Both DSLs and GASM can be used like a SLOC to measure work performed at the Systems level
- In each case we are looking for a complete model so that the requirements coherence can be checked through the coherence of the GASM model or the DSL Design Model
- Model is a synthesis first, not a simulation support. Simulations can come later when the design stabilizes
- The model is a support for thinking about the system that is being built