Proposed "Integral Systems Engineering Methodology" Architectural Design Language

BASED ON THE EXISTING "INTEGRAL SOFTWARE ENGINEERING METHODOLOGY" (ISEM) ARCHITECTURAL DESIGN LANGUAGE FROM WILD SOFTWARE META-SYSTEMS.

RESPONSE TO REQUEST FOR INFORMATION(SE DSIG RFI 1)FROM THE SYSTEMS ENGINEERING DOMAIN SPECIAL INTEREST GROUP(SE DSIG)OF THE OMG

Kent D. Palmer, Ph.D.

P.O. Box 1632
Orange CA 92856 USA
714-633-9508
palmer@exo.com, kent@palmer.name

Copyright 2002 K.D. Palmer.
All Rights Reserved. Not for distribution.
Version 0.02; 8/9/02; uml01a02.doc

Keywords: UML, Systems Engineering, Architectural Design Languages, ISEM, Systems Theory, Meta-systems

Introduction

This is a response to the RFI of DSIG on Systems Engineering use of UML. In this response the specific questions of the RFI will not be answered but more general or foundation questions that should be considered will be brought up for the consideration of the DSIG.

The author of this response is a working Systems Engineer and formerly a Software Engineer of a major aerospace company. In conjunction with doing design work on several systems both at the software and systems engineering levels the author has engaged in an independent program of research into the basis of design methods and the connection between architectural design and systems theory. The results of the first stage of this research is to be found in the author's electronic book called Wild Software Meta-systems[1]. The author is now engaged in a Ph.D. program[2] in Systems Engineering which in part seeks to rework this research into software methods and design languages into a design language which will be of use at the systems architectural design level as well as the software architectural design level. Preliminary results of this new research can be seen in a paper presented at INCOSE 2002[3].

In this paper I would like describe how the previous work on the Integral Software Engineering Methodology architectural design language may be relevant to the committees work on a version of UML for Systems Engineering.

UML and Methodology

UML purports to be a visual language for use in designing software systems which

[1] http://archonic.net/wsms.htm

[2] See http://www.seec.unisa.edu.au/

[3] See http://archonic.net See Vajra Logic and Mathematical Meta-models for Meta-systems Engineering and other papers on Meta-systems Engineering.

brings together a set of various diagrams found useful in representing various aspects of software systems requirements, architecture and design. It sidesteps completely the question of methodology and avers that its diagrams might be used by a variety of methodologies. In my own research I have concentrated on the question as to where design methods come from and secondarily what is the nature of the design languages that should be used as representations by those methods. This is one of the great unsolved problems of Software Engineering. There is a rush to produce an infinite variety of methods without sufficient attention being paid to the foundation of those methods. I have attempted to ground methods by an appeal to Systems Theory especially that proposed by George Klir in Architecture of Systems Problem Solving. In my paper "Software Engineering Design Methodologies and General Systems Theory[4]" which is the first chapter in Wild Software Meta-systems I have attempted to set down the results of my research into the origin of methods and their systematic relations.

Briefly, I believe there are at least four viewpoints on any realtime system which relate to Agent, Function, Data and Event. The minimal methods that correspond to diagrams in UML are bridges between these viewpoints that constitute a field. UML due to some of its assumptions is a skewed representation of this field which is fairly complete. However, UML is a hodge podge of minimal method representations that should be regularized and attached to the underlying field which is organized in relation to the fundamental viewpoints rather than continuing to be free standing, because by not connecting the UML diagrams to the field generated by the viewpoints produces anomalous distortions in the modeling diagrams in relation to the underlying system of minimal methods. I think that as the level of abstraction is raised from Software to Systems Engineering these distortions will become even more telling.

Once a theory had been produced that allowed the relations between the various minimal methods to be determined then it was possible to create a language that attempted to capture the showing and hiding relations that appertained between the various elements that appear in the minimal methods (diagrams). Instead of a visual diagramming representation a design language called Integral Systems Engineering Methodology (ISEM) was formulated. It attempted to capture every relevant fact concerning design architectures. Because it was based on Systems Theory it is also directly relevant to Systems Engineering level architectures of the kind identified by Eberhardt Rechtin in Systems Architecting. My research has shown that exactly the same principles underlay Software and Systems level archtecting. It is just a matter of abstraction what separates the two kinds of architecture. And thus the same minimal methods apply to both levels of abstraction equally. Thus a UML that covers both should be possible. Many of the changes that the Systems Engineering community would like to make in the current version of UML (1.4) are due to the assumptions of the priority of Object Oriented Design over Functional Design built into the language, and because of its specialization to software. In other words the language needs to be made more general and its assumptions which skew it toward object oriented design rather than functional design needs to become more even handed in order to make the language more useful at the Systems Engineering level of abstraction. But by connecting UML as a visual design representation to the underlying methodical field helps to justify these changes, because it gives a motivating theory for the relation of the representations to each other. Otherwise one is merely arguing from

[4] International Journal of General Systems - Vol. 24, No. 1-2, 1996, pp. 43-94

case by case experience of what seems to work or not in particular situations, rather than arguing from a methodological theory grounded in systems theory.

One point of particular interest is that ISEM is not visual but rather a design language representation that is text based. This was done because it was found that visual representations although helpful are in fact semantically weak. As system complexity increases visual representations break down. Thus a design language was created with a rational or regularized and simplified grammar in order to express the complex and multi-way relations between architectural design elements that appeared in the minimal methods. I believe that UML should be extended to have both textual and visual complementary representations. This will allow visualization in the cases where systems representations are simple enough to allow them but will allow complex systems that do not lend themselves to easy visualization to also be represented when necessary. A design language is unlike a formal language of the formal methods parlance or programming languages in that it is intrinsically open to the addition of new elements and new statements about elements to be added as needed by the designer on the fly. Such languages are very difficult to formally define so that they can be parsed. But in practice it is very easy to make up such languages as needed augmenting the descriptive capability of the minimal method language in the process of design itself. Some technique for extending the language so that it might be parsed after the fact needs to be developed. We should be able to design either visually or textually and then see the results in the other mode. That is a challenge that needs to be met by tool designers. I believe that textual design representations will become even more important at the Systems Engineering level abstraction, where representations will increase in complexity and synergy and confront the need for more domain specific modifications.

I am in the process of attempting to redesign the ISEM language to function at the Systems Engineering level of abstraction in such a way that it will be useful for both software real-time system architectonic and also for systems level architectonic. Part of that redesign will be to bring the minimal languages into compliance with UML 2.0 giving an alternative to visual representation for those who need more robust modeling capability. It is strange that we use programming languages or formal method languages to describe systems but insist on visualization for design. One would think that it would be natural for us to use mini-method based languages to describe systems designs. However, one of the assumptions built into UML is that visualization of design is the only way to go. This means that many diagrams might be necessary to describe a very complex system when a more simple and synergetic representation might be made in a few well crafted design language statements. At the Systems Engineering level of abstraction this sort of synergistic simplification of representations will be even more important especially if we also describe the system of systems level of abstraction as well.

Here I would like to point out some features of ISEM that do not appear in UML that should be considered for addition into UML 2 and beyond to make it more useful for System Engineering. The key to the ISEM language is the entity relation diagram for the language[5]. In these diagrams the various key nouns in the language, design elements, are represented as boxes and the lines between these boxes represent statements in the language. These diagrams were used to attempt to assure completeness in the ISEM language. On page 836 the category SET is defined and on the next page 837 the category LIST is defined along with their various statements. ISEM is a series of sub-

[5] http://dialog.net:85/wildsoft/wilds11b.pdf

languages that are directed at various minimal methods (diagrams in UML). But the language starts with SET and LIST. Recently in a paper[6] presented to the INCOSE 2002 conference the foundations of ISEM were augmented by extending to MASS and MIXTURES on the basis of the realization that we have both count and non-count approaches to describing things in our language. To classical logic pervasion logics must be added to describe the interrelation of masses and their instances in a way similar to the description of sets and the particulars that they contain. In the new version of ISEM it is intended that the entire language will contain this duality between sets and masses. Masses are used to define the emergent level of executing systems which display supervenient properties. Thus we design in terms of SET but we experience executing systems with their emergent properties mostly in terms of MASS. Thus going back and forth between SET and MASS views will allow us better ways of describing the as built systems in relation to designs. Fortunately there is a logics related to both of these categories, SET and MASS so that we can reason about our designs with a normal logic but we can also reason about the emergent system in terms of a pervasion logic such as those developed in China and India and which we know as G. Spencer Brown's Laws of Form.

Lists are combinations of Mass and Set properties, as are Mixtures. Lists let in identity under the auspices of difference endemic to the Set. Mixtures let in difference under the auspices of identity endemic to the Mass. This duality between set and mass needs to be maintained throughout the building of the minimal method languages on top of the mathematical categories of Set and Mass. Unfortunately in mathematics it is not recognized that the Anti-Set is a Mass and it is considered to be merely exactly the same with its arrows reversed in category theory descriptions. Instead we need to recognize that a genuine duality is built into the mathematical foundations of our architectural representations and exploit that in order to be able to represent the difference between designs and the emergent properties of running systems. This is especially important for validating systems.

On page 838 the System sub-language is described and there we see the Meta-system as the dual of the System. Quite a bit of theoretical work has been done by the author on the nature of Meta-systems as the dual inverse of Systems. See the INCOSE 2000 paper on Meta-Systems Engineering. This duality between systems and meta-systems needs to be reflected in the language and in the updated version of the language the meta-systems theory results will be incorporated in order to strengthen the description of meta-systems in the language.

On page 839 the difference between the infrastructure and the hierarchy can be seen in relation to the various levels of service that need to be defined in order to have a good way of describing the architecture. These include platform, implementation, backplane integrator, service and application. This is like the levels in the standard communications layer model.

On page 840 we see how each level of the generic hierarchy is tied into the grid which is made up of four orthogonal leveling schemas called: tier, layer, partition and strata. The grid is a mechanism taken from another scholar[7] who studied how very complex systems are represented and who attempted to describe a composite method for describing such system. Using the grid it is possible to describe any architecture as a set of orthogonal levels which respond to different viewpoints generated by the concerns of the customers and other

[6] op cit Vajra Logic paper

[7] unfortunately I have lost this reference

stakeholders. The grid is multi-dimensional orthogonal division of the architecture according to different viewpoints. ISEM has four such division schemes built in but any number can be added. Notice that both the System and the Meta-systems have grids. The grid is a technique that should be added into UML as a means of organizing very complex systems.

On page 841 notice that the Architecture is composed of on the one hand the levels of service and on the other hand the fundamental orthogonal divisions which define intersections. Architectural elements appear at the intersections of these various viewpoints on the architecture. Levels of service and along with the system, meta-system and environmental grids make up a domain. As we see on page 843 the domain is the gateway to the description of the packaging of the system and meta-system in terms of configuration items and products that describe the system. The domain also organizes the requirements. I support the use of Gurevich Abstract State Machines[8] for the operational description of requirements for both systems and meta-systems. On page 845 we see that Domains are comprised category systems. They are related to disciplines.

On page 846 we can see that an architecture is composed of node and arc networks that appear together at intersections of the of the coordinate systems that are projected by the various viewpoints. On page 847 we see that these nodes and arcs are category theory abstractions for the various specific kinds of elements and lines between elements that appear on the various minimal method diagrams. The language goes on to describe each of these mini-design languages given a minimal set of statements that will allow these various UML like diagrams to be described. Some interesting variations with respect to well know diagrams might be gleaned from these descriptions. The minimal methods themselves are described in the appendix Description of Software Design Minimal Methods[9].

For the most part this entire language can be seen as applying to the System and Meta-system levels of Architectural Design as well as the Software level because it was based on advanced Systems Theory from the beginning. The same methods seen in UML are found in this language for the most part. What is different is the fact that it is based on systems theory and it describes the genesis of minimal methods which are more than just ad hoc modeling diagrams but have a substantive theory as their basis. But there are many nuances to the representations of these mini-design languages that should be considered for inclusion into any UML for Systems Engineering as a refinement, especially those aspects that are directed at a systems level design and which gives increased flexibility and robustness to the representations of systems architectures. A lot of research needs to be done into the foundations of Systems Engineering in systems theory and the best expression of systems engineering methods for architectural design. The production of ad hoc diagramming models for visualization is a short cut which industry has taken but there is very little foundation theoretical work to back up the selection of these design elements and these relations in the various diagrams. More work needs to be done to supply those foundations and validate the usefulness of particular representations and combinations of representations in the context of methodologies (which I define as sequences of the use of minimal methods). The motivation of ISEM was to study explicitly those design elements that have showing and hiding relations in various minimal methods (diagrams) from different viewpoints. It was not meant to be a design language as such. But it could serve as such

[8] See Gurevich Abstract State Machines in Theory and Practice at http://archonic.net

[9] http://dialog.net:85/wildsoft/wilds10b.pdf

a design language to augment the visualization characteristics of UML when applied to more complex systems, or systems where greater synergy of representations is necessary, or systems where compactness of representation is for some reason valued, or for systems in which we want to reason about our designs in ways that is difficult to do with diagrams.

I offer ISEM to the DSIG on Systems Engineering and UML as an alternative and perhaps contrarian view of what design languages should be like in hopes that some unique aspects of ISEM might spark more debate as to how Systems Architectures should ultimately be represented.

Further Research Work

Part of my research for my dissertation at SEEC will be upgrading the ISEM language for application to the Systems Engineering level of Abstraction in order to help define better ways of representing design architectures of Systems and Meta-systems. This forms a practical side to this work on a thesis called The Foundations of Emergent Meta-systems Theory and Practice. In this I hope to show that combining Set and Mass ways of looking at things along with their associated logics is valuable for Systems architectural design. It will be see what happens when the Set and Mass approaches are applied to the minimal methods themselves. It will also be interesting to see what happens when the language is extended to more concretely describe meta-systems as well as systems. The language is meant to be a test bed for trying out alternative representations of architectures. As such it complements UML 2.0 which is meant to be a working language rather than a basis of theoretical explorations of the limits of architectural representation. Hopefully if this work goes forward as planned it might be a basis for contributing to UML 3.0 in which all the needs of Systems Engineers are not just addressed but firmly founded in Systems theory and in which Methodological considerations have been fully explored.

Systems Engineers like Software Engineers are not normally Methodologists or Systems Theorists and cannot be expected to do the foundational work needed to shore up their working methods and representational concepts. But this is essential work that needs to be done by the nascent academic branch of Systems Engineering. Spot cures sometime turn out to be dangerous and deadly in the long run. There is a big difference between proposing new representations and showing their efficacy. It is easy to do the former but very difficult to do the latter. But ultimately if we want our systems to be safe, secure, and have various other x-ilities then we will want that substantiation of our representations and methodologies for designing and building systems. So although I applaud the attempt to get a UML that Systems Engineers can use as quickly as possible, I would like it to be recognized that this is just the beginning of a long road of research which validates the actual usefulness of UML as a means of representing Systems and Meta-systems beyond the hype of tool vendors or champion practitioners. We need to propose many new minimal methods and methodolgical combinations of minimal methods and try them out in various circumstances in order to be sure that the ones we have chosen to be incorporated into the language are the best. Are we going to jump to a point solution without generating the alternatives in our design description languages when we advocate not doing that in our systems engineering design practice? ISEM is an outlier text based alternative design representation that deserves to be considered, and also evolved in order to balance the dominance of the visually oriented language of UML. This needs to be done if for no other reason than to allow us to say we have genuinely explored other possible design representations in our search for the best way

to represent systems architectural designs.

ABOUT THE AUTHOR

Kent Palmer[10] is a Principal Systems Engineer at a major Aerospace Systems Company. He has a Ph.D. in Sociology concentrating on the Philosophy of Science from the London School of Economics and a B.A. in Sociology from the University of Kansas. His dissertation on The Structure of Theoretical Systems in Relation to Emergence[11] focused on how new things come into existence within the Western Philosophical and Scientific worldview. He has written extensively on the roots of the Western Worldview in his electronic book The Fragmentation of Being and the Path Beyond the Void[12]. He has had nearly twenty years experience[13] in Software Engineering and Systems Engineering disciplines at major aerospace companies based in Orange County, CA. He served several years as the chairman of a Software Engineering Process Group and is now engaged in Systems Engineering Process improvement based on CMMI. He has presented a tutorial on "Advanced Process Architectures[14]" which concerned engineering wide process improvements both in software and systems engineering. Besides process experience, he has recently been the software team lead on a Satellite Paybad project and a systems engineer on a Satellite Ground System project. He has also engaged in independent research in Systems Theory which has resulted in a book of working papers called Reflexive Autopoietic Systems Theory[15]. A new introduction to this work now exists. It is called *Reflexive Autopoietic Dissipative Special Systems Theory*[16]. He has given a tutorial[17] on "Meta-systems Engineering" to the INCOSE Principles working group. A paper with this title was also published in the INCOSE 2000 proceedings. Further papers on this subject were published in the INCOSE 2002 proceedings[18]. He has written a series of papers on *Software Engineering Foundations* which are contained in the book With Software Meta-systems[19]. He has taught a course in "Software Requirements and Design Methodologies" at the University California Irvine Extension. He is a Ph.D. candidate at the Systems Engineering Evaluation Center of the University of South Australia with a proposed dissertation title of The Foundations of Emergent Meta-systems Theory and Practice.

[10] Palmer@exo.com, palmer@dialog.net, palmer@interpentrating.net

[11] http://dialog.net:85/homepage/disab.html You man also try http://dialog.net or http://think.net or http://archonic.net for any of the web related material.

[12] http://dialog.net:85/homepage/fbpath.htm

[13] http://dialog.net:85/homepage/resume.html

[14] http://dialog.net:85/homepage/advanced.htm

[15] http://dialog.net:85/homepage/refauto2.htm

[16] http://dialog.net:85/homepage/autopoiesis.html

[17] http://dialog.net:85/homepage/incosewg/index.htm

[18] http://archonic.net

[19] http://dialog.net:85/homepage/wsms.htm