2005

Toward Practical Measures of Complexity in Real Time Modeling Methods

John Erickson
University of Nebraska at Omaha, johnerickson@mail.unomaha.edu

Keng Siau
University of Nebraska-Lincoln, siauk@mst.edu

Toward Practical Measures of Complexity in Real-Time Modeling Methods

John Erickson
University of Nebraska at Omaha
johnerickson@mail.unomaha.edu

Keng Siau
University of Nebraska-Lincoln
ksiau@unlnotes.unl.edu

ABSTRACT

Systems development methods have become more complex, concurrently with many of today's systems. UML has been criticized for its complexity. Using Rossi and Brinkkemper's (1996) complexity metrics, Siau and Cao, (2001) completed a complexity analysis of UML and other modeling techniques, finding that UML is more complex than other techniques. Siau, Erickson and Lee (2002) proposed that Rossi and Brinkkemper's metrics present the theoretical maximum complexity, as opposed to a practitioner-based complexity, that must be less than the theoretical maximum. Erickson and Siau (2004) proposed that a subset of UML (a kernel) composed of the most commonly used constructs, could be equated with the complexity that practitioners face when using UML. The current research extends Erickson and Siau's (2004) work by analyzing UML complexity in the specific genre of real time systems. A Delphi study was conducted using expert practitioners, in an attempt to identify a use-based real-time UML kernel.

Keywords

UML, complexity, complexity metrics, Delphi study, modeling method metrics, Real-Time systems.

INTRODUCTION

Our world becomes more complex every day. Technology and system developers promise a reduction in life's everyday burdens because of technological advances. While perhaps some of that has proved real, systems have expanded to encompass more and more of the tasks once performed manually. Naturally, then it should come as no great surprise that systems are now more complex than ever. In addition to added complexity, we also expect our systems to respond in real time to meet our needs.

If systems and software have become more complex, then is not out of line to suppose that the underlying systems development process has become more complex as well. Further, it should be reasonable to presume that specific development methods have paralleled our seemingly inexorable march toward greater and greater complexity. An example is the soon-to-be-released UML (Unified Modeling Language) 2.0. While the original UML 1.X has been criticized for its complexity, inconsistent semantics, and ambiguity (Dobing and Parsons, 2000; 2002; Zendler, Pfeiffer, Eicks, and Lehner, 2001), the early version also lacked truly useful extension mechanisms that would facilitate its use in a variety of settings. Thus, UML 2.0, purportedly addresses the semantics, ambiguity and extension issues, but likely at the expense of additional complexity, since it will include four new diagram types along with their related constructs.

This research is aimed at developing a means to deal with complexity, by developing means to measure practical complexity, specifically in a real-time environment. A practitioner-based Real-Time UML kernel is presented as the results of a Delphi study. The application of the results to real-time environments, using a metrical analysis technique is in process currently, so this paper represents a research in process.

Literature

Rossi and Brinkkemper's (1996) research proposed and developed a relatively easy to use and straightforward means to quantitatively measure system development methods. Specifically, the metrics are based on metamodel techniques, and purport to measure the complexity of the method under analysis. According to Rossi and Brinkkemper (1996), complexity is critical to measure because researchers supposed complexity to be closely related to how easy a specific method is to use, and also how easy the method is to learn.

Siau and Cao's (2001) research applied Rossi and Brinkkemper's complexity metrics to the Unified Modeling Language (UML). Their reasoning for using the particular metric set is that they contend that the metrics are among the most comprehensive, and that Rossi and Brinkkemper's approach "…have been used to evaluate the ease-of-use of OO techniques." Siau and Cao (2001) also compared UML's complexity with 36 OO techniques from 14 methods, as well as each of the 14 methods in aggregate.

One of Siau and Cao's (2001) noteworthy findings is that UML is far more complex (from 2 to 11 times more complex) in aggregate than any of the other 13 methods. The relative overall complexity highlights one of the issues regarding UML, with the result that it can appear overwhelming to those new to UML (Figure 1). Additionally, when human cognitive limitations to short term memory are added to this mix, UML can appear even more difficult to master.

Diagram	**N (O_T)**	**n (R_T)**	**N (P_T)**	$\overline{P}_O$ **(M_T)**	$\overline{P}_R$ **(M_T)**	$\overline{R}_O$**(M_T)**	$\overline{C}$ **(M_T)**	**C`(M_T)**
Class	7	18	18	1.71	1.22	2.57	0.1	26.40
Use Case	6	6	6	1	0.83	1	0.17	10.39
Activity	8	5	6	0.75	0.2	0.63	0.13	11.18
Sequence	6	1	5	0.67	6	0.17	0.13	7.87
Collaboration	4	1	7	1	8	0.25	0.14	8.12
Object	3	1	5	1.67	3	0.33	0.33	5.92
StateChart	10	4	11	1	0.5	0.40	0.09	15.39
Component	8	10	9	1	3.6	1.25	0.11	15.65
Deployment	5	7	5	1	1.14	1.40	0.2	9.95

Figure 1

Legend:
n (O_T) – count of object types per technique.
n (R_T) – count of relationship types per technique.
n (P_T) – count of property types per technique.
$\overline{P}_O$ (M_T) – average number of properties for a given object type.
$\overline{P}_R$ (M_T) – average number of properties per relationship type.
$\overline{R}_O$(M_T) – number of relationship types that can be connected to a certain object type.
$\overline{C}$ (M_T) – average complexity for the entire technique.
C`(M_T) – total conceptual complexity of the technique.

Theoretical Basis

Humans generally have cognitive problems processing information that is overly complex (Anderson and Lebiere, 1998; Miller, 1956). This problem surfaces often as people build information systems, which tend to be extremely complex. ACT breaks knowledge into two parts, declarative knowledge and procedural knowledge. Declarative knowledge is similar to that knowledge captured in an encyclopedia or dictionary – it is a list of what we know. Procedural knowledge, on the other hand is knowledge about how things work. Procedural knowledge depends on declarative knowledge as a starting point, but uses that knowledge to help solve problems (Anderson and Lebiere, 1998). Declarative knowledge is produced in chunks, and is constrained by our cognitive limits (Miller, 1956). Procedural knowledge is used to create production rules that describe productions, or specific steps we use to solve common and complex problems (Anderson and Lebiere, 1998). In short, ACT tells us that the more complex a problem is, the longer it will take us to process the problem, and come up with a solution it (Figure 1).

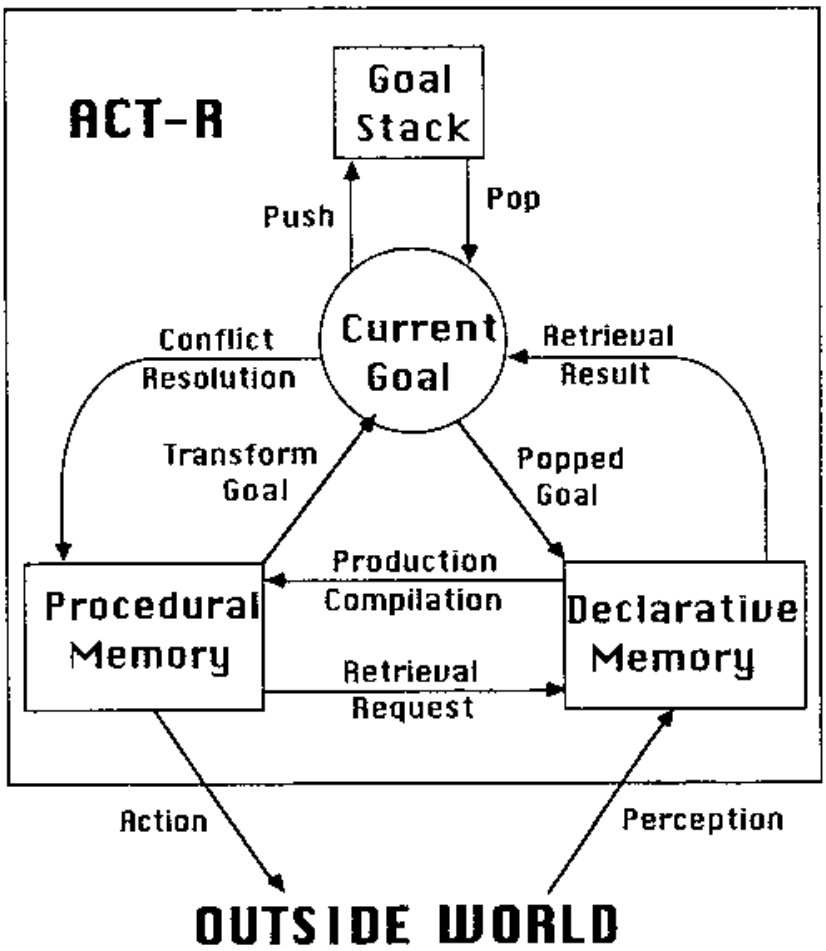

Figure 2 (Source Anderson and Lebiere, 1998)

Complexity, in the context of this research, can take many forms. For the purposes of this research, complexity will be approached from two separate but closely related perspectives; cognitive complexity as related to human perception, and structural complexity, as related to the structural properties of the diagramming techniques found in modeling approaches such as UML diagrams. In this context cognitive complexity can be defined as the mental burden people face as they work with systems development constructs.

Further, in addition to the theoretical basis of cognitive complexity detailed in the literature review, the research proposes to adopt the ideas on the definition of structural complexity as proposed by Briand, Wüst, and Lounis (1999), in which the physical (structural) complexity of diagrams affects the cognitive complexity faced by the humans using the diagrams as aids to understand and/or develop systems. (See Figure 3 below)

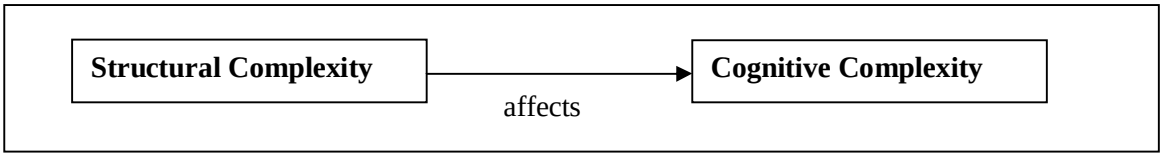

Figure 3 (Adapted from Briand, Wüst, and Lounis, 1999)

Since cognitive complexity, as defined for this research, is difficult and arguably even impossible to measure, structural complexity will be used to explain cognitive complexity. Structural complexity can be defined as a function of the number of distinct elements (or constructs) that constitute a given diagramming technique. Rossi and Brinkkemper (1996) formulated seventeen distinct definitions relating to the structural complexity of each diagramming technique. Using all available constructs (the metamodel component); these definitions form an estimate of the total structural complexity of the diagramming technique, which this research terms theoretical complexity.

Structural complexity is a part of the structural characteristics of the information or modeling system, and for this research refers to the elements, or constructs that comprise a given diagramming technique. These constructs would include meta-construct types such as objects (classes, and interfaces), properties (class names, attributes, methods, and roles), and relationships and associations (aggregations generalizations, specializations).

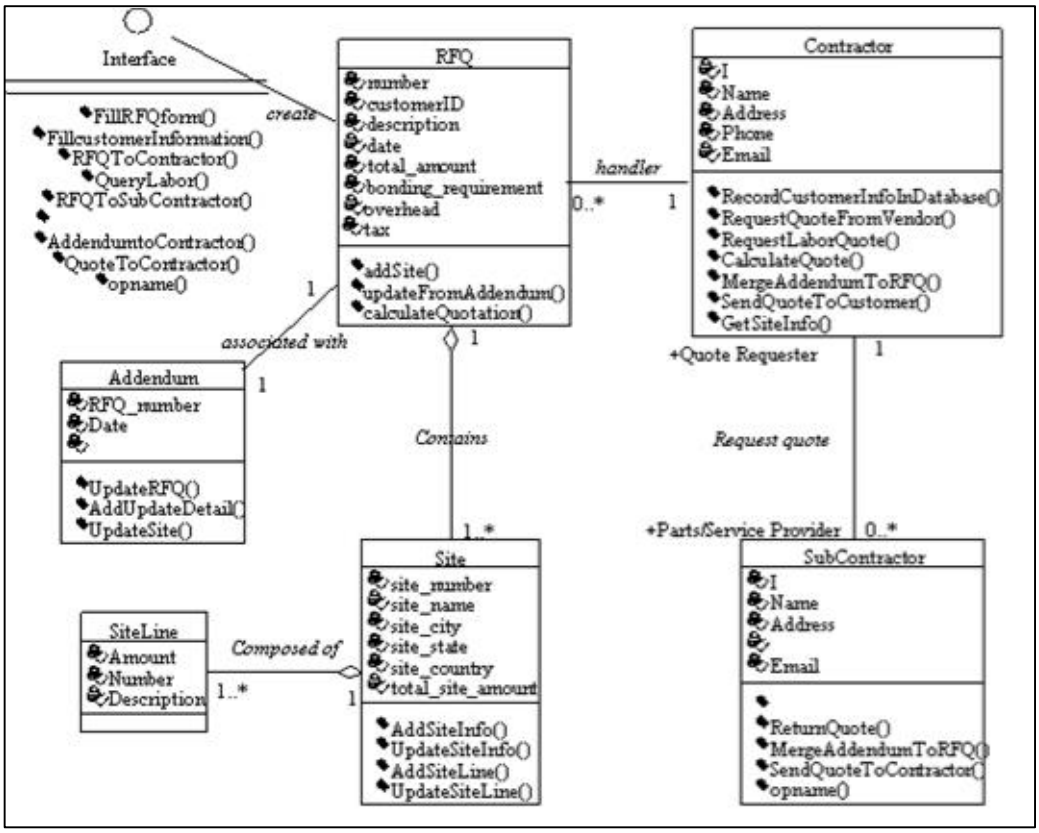

Figure 4 (Erickson and Siau, 2003)

For example, the class diagram in Figure 4 above uses a number of constructs, in presenting the information it represents. There are six classes, and one interface. The classes each have a name, attributes, and methods (i.e. the RFQ class has eight attributes – number, customerID...tax; three methods – addSite(), updateFromAddendum(), and calculateQuotation(). In addition, the class has relationships with other classes (Contractor, Site, and Addendum), and also with the (Customer) Interface. However, the diagram does not make use of all the constructs that could be used in a class diagram. The Rossi and Brinkkemper metrics serve as the operational definition (as well as a measure) of the structural complexity of diagrams, which can be equated with theoretical complexity. The current research uses Erickson and Siau's (2003) definition of practical complexity, as a subset of theoretical complexity, to asses the complexity of real-time systems from a UML perspective.

Siau et al (2002) provided some evidence indicating that the theoretical metrics (total structural complexity) do not adequately capture the complexity practitioners face when using UML class and use case diagrams. In addition, Erickson and Siau (2004) provided further support for the idea of practical complexity by conducting a Delphi study aimed at identifying a kernel of UML that was based on developer perceptions of the utility of the various constructs and diagrams comprising UML. These results motivate the research question:

Can an alternative set of complexity measures, as defined by Erickson and Siau (2004), be developed for specific application areas, in particular, Real-Time UML?

The research objectives for this study are, (i) using the Erickson and Siau (2004) results determine the most commonly used constructs in the UML diagrams for real-time applications and, (ii) use the results to define or identify an alternative to the established metrics, and propose that they represent a real-time kernel of UML.

Method

The research investigates practical complexity, and the formulation of a UML kernel by means of a Delphi study. Delphi studies attempt to form a reliable consensus of a group of experts in a specialized area (Ludwig, 1997). The approach is a process that focuses on collecting information from the expert group though a series of questionnaires, and providing feedback to the group between questionnaires. Usually the group of experts is geographically dispersed, as will be the case for many of the subjects participating in this research (i.e. the different companies and people involved). The questionnaires are usually designed to allow the collection of expert opinions on the subject, and then to facilitate the refinement or focus of the subsequent versions to narrow in on a consensus.

The subjects were asked to respond to their use of the standard UML constructs in a real-time application as well as to the Real-Time extension for UML as identified below. The Real-Time constructs used for this research were identified from Real-Time Application Extension for UML (Douglass, 1999), and the Object Management Group (OMG) Web site (2003).

Class types: Clock, Physical Time, Physical Instant, Duration, Time Value, Time Interval, Resource Instance, Timing Mechanism, Timer, Clock Interrupt, Timed Event, Timeout, Stimulus, Stimulus Generation, Timed Stimulus, Event Occurrence, Scenario, Timed Action, Delay, Resource Manager, Time Service,

Association types: Ordered, Measurement, Reference Clock, Current Value, Maximal Value, Resolution, Offset, Accuracy, Generated Interrupts, Generated Timeouts, Timestamp, Origin, Cause, Effect, Time, Start, End, Duration, Physical Instant, Measurement, Physical Time, Clock Tick, Time Interval, Reference Clock, Generated Events, Timing Mechanism, Clock, Timer

Properties: current value, reference clock, origin, maximal time value (offered QoS characteristic), resolution (an offered QoS characteristic), stability (an offered QoS characteristic), skew (an offered QoS characteristic), drift (an offered QoS characteristic)

Standard services: set time (the interpretation of this is mechanisms specific), get time (the interpretation of this value is specific to each mechanism), reset (which is also specific to each mechanism), pause, start

Results

Based on respondent ratings, the final order of importance (with the mean rating score in parentheses) was as follows: Class (1.23), Statechart (1.31), Sequence (1.46), Use Case (1.92), Component (2.00), Activity (2.11), Deployment (2.38), Object (2.91), and Collaboration (3.10). The top 4 diagrams in terms of mean rating scores were also selected by the respondents to be included in the kernel, at respective consensus levels of 100%, 92.3%, 100%, and 100%. The next highest rated diagram in terms of importance attained only a 23% consensus level (although the sixth highest rated diagram, Activity, was rated at 53.8% for inclusion in the kernel), indicating the respondents clearly distinguished between important and less important diagrams, and kernel and non-kernel diagrams. Figure 5 below provides summary results. Results are available for all nine UML diagrams, but are not presented here because of space limitations.

Construct	Mean	Standard Deviation	% "Yes" for Kernel
Class	1.23	0.60	100.0%
Statechart	1.31	0.48	92.3%
Sequence	1.46	0.52	100.0%
Use Case	1.92	0.86	100.0%
Component	2.00	0.82	23.1%
Activity	2.11	0.33	53.8%
Deployment	2.38	0.74	0.0%
Object	2.91	1.04	0.0%
Collaboration	3.10	0.88	0.0%

Figure 5 UML Real-Time Diagram results

Figure 6 below illustrates the respondent results for the UML Real-Time extension. There appears to be a relatively clear break between the kernel constructs and the non-kernel constructs that closely corresponds with the importance ratings as well.

CONCLUSION

In contrast to Erickson and Siau's 2004 results, this research identifies a slightly different UML kernel, one that likely reflects the needs of applications operating in real-time environments. However, it is also noteworthy that the kernel differences are relatively small as well, since the kernels identified are identical in both cases, except for the order of the diagrams.

Erickson and Siau's 2004 results also indicate that a smaller number of constructs may be an indicator of less complexity. They used the Rossi and Brinkkemper metrics to provide a more concrete analysis of the kernel, with the results that the practical-based UML kernel reflected more than a 29% decrease in complexity. A similar analysis is currently in progress for the real-time results, and should be available by the conference date. A comparison with the Erickson and Siau (2004) results should provide additional insight into the complexity of UML.

Construct	Mean	SD	%Y
Clock	1.167	0.408	1.000
Timeout	1.167	0.408	1.000
Timer	1.167	0.408	1.000
Time	1.200	0.447	1.000
Event occurrence	1.400	0.548	0.857
Scenario	1.400	0.894	1.000
Time Value	1.600	0.548	0.857
Timed action	1.833	1.602	1.000
Timed event	2.167	1.472	0.143
Time interval	2.200	0.837	0.000
Physical time	2.286	1.496	0.143
Pause	2.333	1.633	0.143
End	2.400	1.673	0.429
Start	2.400	1.673	0.429
Stimulus	2.400	1.673	0.143
Cause	2.500	1.517	0.286
Clock interrupt	2.500	1.378	0.143
Delay	2.500	1.378	0.000
Generated interrupts	2.500	1.378	0.143
Timing mechanism	2.500	1.378	0.286
Duration	2.571	1.512	0.143
Resource instance	2.600	0.548	0.143
Generated timeouts	2.667	1.366	0.143
Get time**	2.800	1.483	0.143
Set time**	2.800	1.483	0.143
Time service	2.800	1.483	0.143
Timestamp	2.800	1.304	0.143
Effect	2.833	1.329	0.143
Physical instant	2.857	1.215	0.000
Current value	3.000	1.225	0.000
Measurement	3.000	1.225	0.143
Ordered	3.000	1.225	0.143
Reference clock	3.000	1.414	0.000
Reset**	3.000	1.225	0.143
Resource manager	3.000	1.225	0.143
Accuracy	3.200	1.095	0.000
Clock tick	3.200	1.095	0.143
Drift	3.200	1.095	0.143
Generated events	3.200	1.095	0.143
Offset	3.200	1.095	0.000
Resolution	3.200	1.095	0.000
Skew	3.200	1.095	0.143
Stability	3.200	1.095	0.000
Stimulus generation	3.250	1.258	0.143
Timed stimulus	3.250	1.258	0.143
Maximal value	3.400	0.894	0.000
Maximal time value*	3.500	1.000	0.000
Origin	3.667	1.155	0.000

Figure 6 UML Real-Time Extension Construct Results

References

1. Anderson, J., and Lebiere, C. (1998) *The Atomic Components of Thought*, Lawrence Erlbaum Associates.
2. Briand, L., Wüst, J., and Lounis, H. (1999c). "A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study." 21st International Conference on software Engineering. Los Angeles, CA. pp 345-354.
3. Dobing, B. and Parsons, J. (2000). Understanding the Role of Use Cases in UML: A Review and Research Agenda. *Journal of Database Management*. Vol. 11. No. 4. pp. 28 – 36.
4. Douglass, B. 2000. *Real-Time UML Second Edition: Developing Efficient Objects for Embedded Systems*, Addison-Wesley.
5. Erickson, J. and Siau, K. (2004) Theoretical and Practical Complexity of Unified Modeling Language: Delphi Study and Metrics Analyses. International Conference on Information Systems. Washington, DC. December.
6. Erickson, J. and Siau, K. (2003). "Unified Modeling Language? The Good, The Bad, and The Ugly." in: Toppi, H., Brown, C. (eds.). IS Management Handbook. Auerbach.
7. Fenton, N. and Pfleeger, S. (1997). *Software Metrics A Rigorous and Practical Approach.* PWS Publishing. pp. 243-278.
8. Kobryn, C. (2002). "What to Expect from UML 2.0". *SD Times*. [Online]. Available: http://www.sdtimes.com/opinions/guestview_048.htm.
9. Ludwig, B. (1997). "Predicting the Future: Have you considered using the Delphi Methodology?" *Extension Journal*. October. Vol. 35. No. 5.
10. OMG (Object Management Group) Web site. (2005). [Online]. Available: [http://www.omg.org/gettingstarted/what_isuml.htm.
11. Miller, G. (1956). "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information." *The Psychological Review*. Vol. 63. No. 2.
12. Rossi, M. and Brinkkemper, S. (1996). "Complexity Metrics for Systems Development Methods and Techniques." *Information System.* , Vol. 21. No. 2. pp. 209-227.
13. Siau, K., and Cao, Q. (2001). "Unified Modeling Language (UML) – A Complexity Analysis." *Journal of Database Management*. Vol. 12. No. 1. 26-34.
14. Siau, K., Erickson, J., and Lee, L. (2002) "Complexity of UML: Theoretical versus Practical Complexity." Workshop on Information Technology and Systems (WITS). Barcelona, Spain, December 16-18.
15. Weyuker, E. (1988) "Evaluating Software Complexity Measures. " IEEE Computer. 0098-5589/88-1357.
16. Zendler, A., Pfeiffer, T., Eicks, M., and Lehner, F. (2001) "Experimental Comparison of Coarse Grained Concepts in UML, OML and TOS. " *Journal of Systems and Software*. Vol. 57. pp 21-30.