On Formal Models and Deriving Metrics for Service-Oriented Architecture

Hardeep Singh
Deptt. of Computer Science and Engg.
G.N.D. University, Amritsar, India
E-mail: hardeep_gndu@rediffmail.com

Rupinder Singh
Deptt. of Information Technology
Adesh Inst. of Engg. & Tech., Faridkot, India
E-mail: rupi_pal@yahoo.com

***Abstract*— Whereas the research in service-oriented architecture (SOA) in general is well-past the early phase, the research in the area of architectural or quality metrics for SOA is still in the early stages. A number of formal models for SOA have been proposed in the literature, and many metrics have been derived from them. This paper serves to project that such a recurrence of formal-model-proposals is not of such a magnitude as to be characterized as multiplicity or excess, and that a certain degree of proliferation is a welcome sign, and will only aid an increased understanding of SOA, rather than impede the same in any way. To support this entire view, this paper discusses the important existing formal models, associated metrics, and demonstrates that these models themselves are a fertile ground to create newer metrics. Thereby, it presents many new metrics, and discusses future research possibilities.**

***Index Terms*— Derive, Service-Oriented Architecture, Metric, Model.**

I. Introduction

Service-oriented architecture (SOA) is an architectural style where systems consist of service users and service providers [1]. A related term, service-oriented computing (SOC), means the computing paradigm that utilizes services as fundamental elements for developing applications [2]. As an analogy, SOC is to SOA, what OOP is to OOAD.
SOA can be much better understood in terms of two basic concepts: layers and binding. Fig. 1 shows the SOA layers or the SOA stack [5, 6, 7]. In static binding (see Fig. 2) the service requesters are bound to provided services at design time, whereas in case of the dynamic, run-time scenario (see Fig. 3), service requesters dynamically discover, select the requisite services from a registry, and bind thereof to selected services.

Whereas the research in service-oriented architecture (SOA) in general is well-past the early phase, the research in the area of architectural or quality metrics for SOA is still in early stages. A number of formal models for SOA have been proposed in the literature, and many metrics have been derived from them. We feel that such a recurrence of formal-model-proposals is not of such a magnitude as to be characterized as multiplicity or excess, and that a certain degree of proliferation is a welcome sign, and will only aid an increased understanding of SOA, rather than impede the same in any way. In fact, a certain multitude re-enforces the belief that architecture is indeed multiple-view based [4]. Also since architecture plays a central role in fulfilling the quality requirements, all such architectural metrics in their varying model-based versions and types provide ample opportunities to quantify an otherwise generally elusive concept of quality. Furthermore, this will provide a large universe of metrics to eventually churn out the most relevant ones, conceptually and empirically. To support this entire view, this paper discusses the important existing formal models, associated metrics, and demonstrates that these models themselves are a fertile ground to create newer metrics. We present many new metrics. Some of these are derived in analogy to metrics of other fields within software engineering, and hence are already in the published and debated research space. Such an exercise contributes to the common conceptual foundation for all ideas in software engineering.

The remaining paper is structured as follows. Sections II-V discuss various existing formal models and associated metrics in brief. Sections VI-VIII propose a variety of metrics derivable from the existing models. Section IX concludes and discusses future research possibilities.

II. The Perepletchikov-Ryan-Frampton-Schmidt Model [8, 19, 10]

In the general case, a service-oriented system *SOS* is formally defined as: SOS =<SI, BPS, C, I, P, H, R>, where SI is a set of all service interfaces in the system; BPS is a set of all business process scripts; C is a set of all OO classes; I is a set of all OO interfaces; P is a set of all procedural packages; and H is a set of all package headers. Generically, the elements of these sets are called service implementation elements, ***e***. Given a system, *SYS,* a service *s* can be defined as: $s = <si_s, BPS_S, C_s, Is, P_s, H_s, R_s>$ is a service of *SYS* if and only if $si_s \in SI \wedge (BPS_s \subseteq BPS \wedge C_s \subseteq C \wedge I_s \subseteq I \wedge P_s \subseteq P \wedge H_s \subseteq H) \wedge (BPS_S \cup C_s \cup I_s \cup P_s \cup H_s <> s)$. Note that <> symbol represents *service membership.* A service boundary is logical rather than physical, thus we need to examine the possible call paths in response to invocations of service operations via the service interface in order to determine whether an element is a

Manuscript received June 22, 2009; revised Jan 14, 2010; accepted Feb 28, 2010.

doi:10.4304/jsw.5.8.866-871

member of a service. Also note that si_s is a singleton set since a service s will have just one service interface si_s. R is the set of overall coupling relationships in a service-oriented design defined on E X E, i.e., R $\subseteq$ E x E, where E is the set of all service implementation element ***e,*** i.e. E= SI ∪BPS∪CUI ∪PU∪ H.

The relationships R can be categorized as:

Incoming relationships, IR(s)

= {(e_1, e_2): $e_1 \in$ (BPS-BPS_s ∪ C-C_s ∪ I - I_s ∪ P - P_s ∪ H - H_s) ∧ $e_2 \in$ (BPS_S ∪ C_s ∪ I_s ∪ P_s ∪ H_s)}

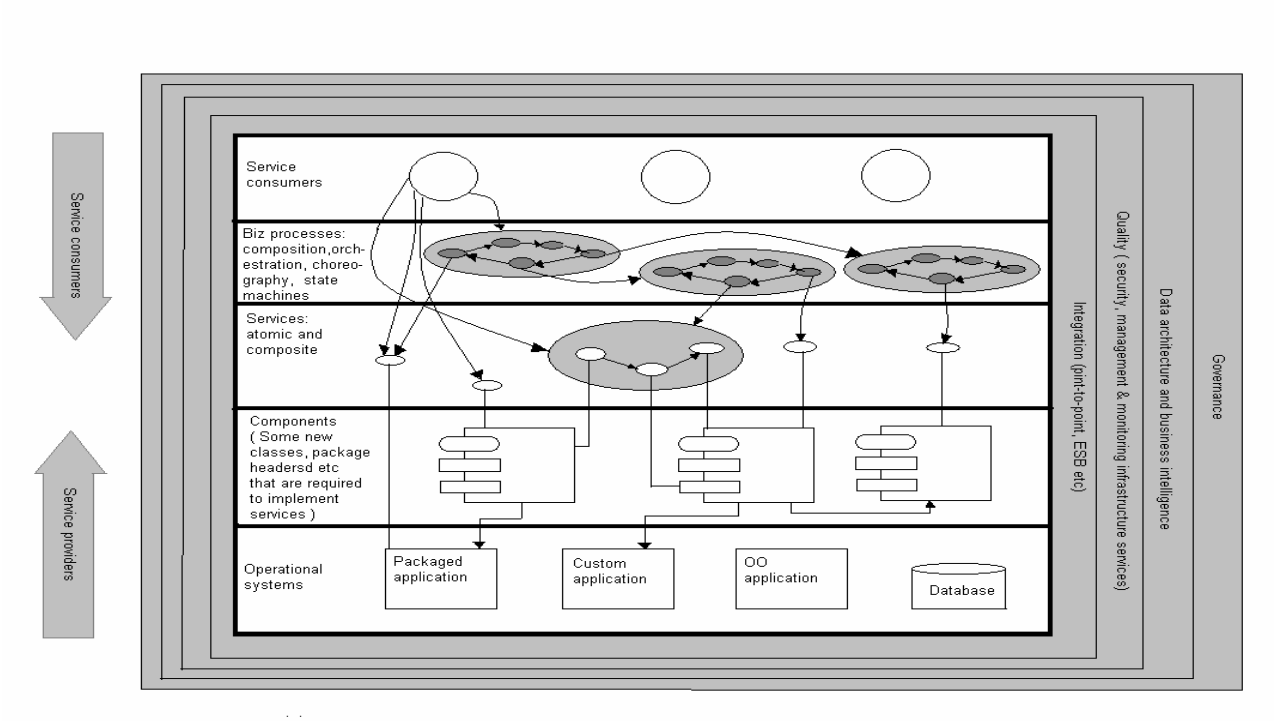

Fig. 1 The SOA Layers

Outgoing relationships, OR(s)

= {(e_1, e_2): $e_1 \in$ (BPS_s ∪ C_s ∪ I_s ∪ P_s ∪ H_s) ∧ $e_2 \in$ (BPS- BPS_S ∪ C- C_S ∪ I - I_s ∪ P - P_s ∪ H - H_s)}

Service incoming relationships, SIR(s) = {(e, si): e ∈ (BPS - BPSs ∪ C - C_S ∪ P - P_s) ∧ si= si_s }

Service outgoing relationships, *SOR*(s) = {(e, si): e ∈ (BPS_S ∪ C_s ∪ Ps) ∧ si ≠ si_s }

Interface to implementation relationships, IIR(s) = {(si, e): si= si_s ∧ e ∈ (BPS_s ∪ C_s ∪ P_s)}

The papers [8, 9, 10] discuss this model and associated metrics in detail. Here we describe one metric from this suite, *Weighted Extra-Service Incoming Coupling of Element (WESICE).*

WESICE for a given service implementation element *e* of a particular service *s* is the weighted count of the number of system elements not belonging to the same service that couple to this element. Formally, WESICE (e) = | {(e, e_1)* WeightFactors: (e, e_1) ∈ IR(s) } | . WeightFactor is a value assigned to different types of relationships based on their perceived influence on system coupling. For example, CP (OO Class → Procedural Package) type of relationship is weighted higher than IC (OO interface→ OO Class) type since the coupling is expected to be 'stronger' in the former case.

III. RUD- SCHMIETENDORF -DUMKE SUITE [11, 12]

A service-oriented system consists of a set of providers' nodes. Each node is providing one or more non-mobile services, each service has one or more operations (business functions). Services can be either atomic or composite, i.e. represent structured collaborations of other services. A few fundamental metrics of the model are:

N, Set of service providers' nodes;

S[n], Set of services provided by the node $n \in N$; and

A, Set of tuples (relation)= *{<InvokerNode, Invoker, ServiceNode, Service, Operation>:* The meaning of each tuple is "Software component (i.e. a service, a composition engine or an end-user GUI application) *Invoker* that is located on the node *InvokerNode* invokes the operation *Operation* of the service *Service* that is located on the node *ServiceNode."}*

The details of this model and associated metrics can be found in [11, 12]. We discuss three metrics from this suite.

AIS[s], Absolute Importance of the Service $s \in S[n]$, $n \in N$ is defined as count of clients which depend on *s*, i.e. which invoke its operations. Note that it does not count here clients that are located on the node *n*.

$AIS[s] = |\pi_{Invoker}\ (\sigma_{InvokerNode \neq n,\ Service=s}\ (A))|$. π is a standard relational-algebra operation, which means projection. σ is another operation meaning selection.

ADS[s], Absolute Dependence of the Service $s \in S[n]$, $n \in N$

is defined as count of other services this service depends on: $ADS[s] = |\pi_{Service}(\sigma_{Invoker=s,\ ServiceNode \neq n}(A))|$.
ACS[s], Absolute Criticality of the Service $s \in S[n]$, $n \in N$ is the product of its absolute importance and absolute dependence: $ACS[s] = AIS[s] \times ADS[s]$.

IV. Liu-Traore Complexity Metric [14]

This metric employs what is called User-system Interaction Elements (USIE) model. For example, a USIE composite service graph is a tree that captures the service hierarchy underlying a particular composite service. We discuss their metric, *Average Service Depth (ASD)*

Given a composite service *c, ASD (c)* = $\sum$ *(NumofServiceDependency (n) / |N|*

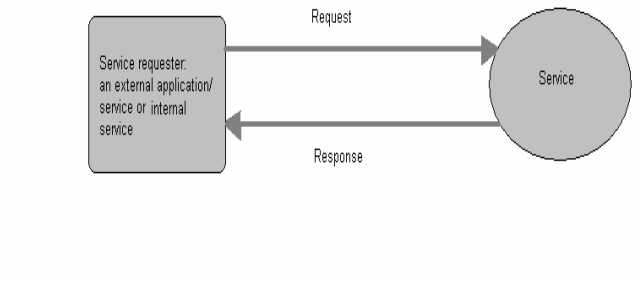

Fig. 2 Static binding

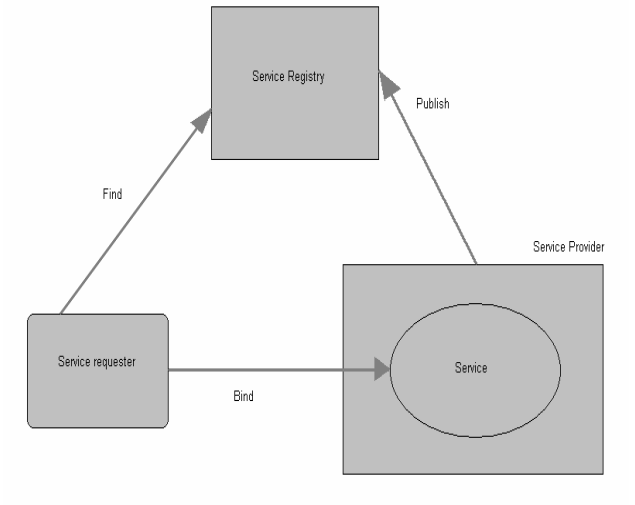

Fig.3 Dynamic binding

Where *n* denotes an atomic service in *c;* $\sum$ runs over $n \in N$; *NumOfServiceDependency (n)* = the number of direct or indirect dependencies of *n; N* is total number of atomic-service nodes in *c*.

The ASD metric actually computes the average number of dependency relationships per atomic service node; therefore, high ASD values indicate high degree of dependencies in the service. Liu and Traore demonstrate in [14] through a case study the empirical relationship between software complexity and attackability, confirming to some extent the widely held belief that complexity has a negative impact on security.

V. Other Models and Metrics

There are a fair number of other models, metrics and measurement work [13-15, 17-22] which we do not treat here in any detail for reasons of space constraints. For example, in [21] Korostelev *et al.* derive a fault model for error detection from the SOA, and introduce metrics to evaluate efficiency of this model.

VI. COHESION METRICS

All proposed metrics are marked ***PM*** (for proposed metric). Van der Hoek et al. [23, 24] have proposed two metrics for PLA (product line architecture). PLA refers to generic architecture for an entire *product family / line* and it explicitly specifies all mandatory components, optional components and variant components (only one of these variant components can be included in a *product instance* of PLA, in other words, a member of product line/ family). Every component in a product instance *provides* some services (in van-der-Hoek model public- access things like operations, methods, functions etc are called a service) and *requires* some service. So, given a product instance, they define two metrics, Provided Service Utilization (PSU) and Required Service Utilization (RSU) as follows:

PSU (X) = (Actual number of provided services of component X utilized within the product instance, P_{actual}) / (total number of services provided by X, P_{total}).
That is, some of the services provided by X may not be utilized in a product instance.
RSU (X) = (Actual number of required services of X available within product instance, R_{actual}) / (total number of services required by X, R_{total}).
The fact that all required services are not available for X in the product may sound counter-intuitive. But, this is acceptable in product line/ family because some of the required services of a component of a product may not be critical to system working, and can be optionally supplied as add-on.
A PSU close to 0 suggests that there is lot of extra functionality in X that is not utilized within the product, i.e. it is"bloated". A value of 1 signifies that it is well utilized. Similarly, RSU close to 0 means it may not function well in products. Further, they derive two metrics CPSU (P) and CRSU (P) for a given product instance.

CPSU (P) = $\sum P_{actual} / \sum P_{total}$ and CRSU (P) = $\sum R_{actual} / \sum R_{total}$.

C stands for compound and the summations run over all components X in the product instance.

We use the formal model of Perepletchikov *et al.* (see Section II). The metrics for SOA can be derived as follows. We consider an ***e*** of service of SOA to be an analogue of component of van der Hoek model and a

service of SOA itself to be analogue of *product instance/ simply product* of van der Hoek model. Also the provided operations of an ***e*** are analogous to provided services of a component and required operations of an **e** are analogous to required services of a component. In fact, this makes sense easily; because an SOA may be some sort of complex PLA wherein a service, whether implemented initially or composed of other services, is indeed a product. Accordingly and analogously, we may define,

PM1: POU (**e**) = (Actual number of provided operations of an **e** utilized within the service) / (total number of operations provided by the **e)**.

PM2: ROU (**e**) = (actual number of required operations of **e** available within service) / total number of operations required by **e)**. (S of van der Hoek model replaced with O)

Then, *C*POU (***PM 3)*** and *C*ROU (***PM4)*** will signify if *an entire service* is more self-contained/ cohesive/ autonomous. Values close to 1 for CPOU and CROU indicate highly autonomous services.

Such metrics may be useful in SOA because SOA is an evolutionary, integrating architecture, wherein disparate legacy components, and new components and external components (external to enterprise) when exposed as services come together, and so it may not be possible to design highly cohesive services from scratch, and it may not be even cost-effective to do so (for example, repeating same type of **e** in various services).

Similarly, we propose *C*POU and *C*ROU for *composite services* (***PM 5*** and ***PM 6***), and in this case we will treat the constituent services as components providing and requiring operations, i.e. either being invoked as subsidiary services or invoking other services. This metric will be of more use when some external services (external to an enterprise) are being used to compose such composite services. Use of external services signifies less cohesive composite services.

VII. COMPLEXITY METRICS

A. Fan-in and Fan-out Variants

Menkhaus and Andrich [25] have proposed a metric suite for embedded software systems. The basic model they use is a directed graph containing "blocks" at vertices, and edges as dependencies. We adapt one of their metrics IOB, instability of block. Blocks that are stable are *both independent and highly responsible*. Blocks are independent if they do not depend upon the results of the other blocks. Blocks are responsible, if changes of this block have a strong and wide-ranged impact upon other blocks. We have simply substituted **e** in place of block. A similar metric, but on an absolute scale has been suggested in [27]. In both [26] and [27], a higher value signifies "instable" or "most easily affected." Actually, these are either variants or direct adaptation of the classic fan-out complexity metric. And all outgoing coupling metrics of Perepletchikov et al [10] (also Section II) are also direct adaptation of fan-out. And so is the ADS of Rud *et al.* (Section III). The IOB differs in that it takes into account fan-in in the denominator. This is factored-in with | IR (***e***) +1 | in the metric below.

The instability of element, IOE (***e***), is defined as

IOE (***e***) = [|OR(***e***)| + |SOR(***e***)|] / [|IR (***e***)| +1+ |OR (***e***)|+ |SOR(***e***)|] e≠ si (***PM 7***)

Where OR(***e***) is a set formed by including the pairs in OR (s), which have occurrence of ***e***; SOR(e) is a set formed by including the pairs in SOR (*s*), which have occurrence of **e;** and IR (***e***) is a set formed by including the pairs in IR (s), which have occurrence of ***e.*** The term 1 in Dr. accounts for incoming coupling, whether direct or indirect, from *si* of *s* to which it belongs. OR (s), SOR (s) and IR (s) are defined as in Perepletchikov formal model discussed in Section II. The scale is [0, 1]. 0 means very less instable, and hence stable, and 1 means highly instable. For the special case of the element *si,*

IOE (*si*) = |IIR(s)| / [| IIR(*s*)| +|SIR(*s*)|] ***e*** = *si* (***PM8***)

IIR(s) and SIR (s) are as defined in the Perepletchikov formal model of Section II.

Zhao et *al.* [17] have suggested a metric, change cost, for what they call service component as an element of service module. It is actually total number of direct and indirect dependencies *upon* a service component, normalized to total number of service components in service module. The authors suggest that the change metric can be calculated with respect to complete SOS. We adapt this later version, and in that sense, *it is complement of the above metric***,** because it signifies the influence or the effect on the rest of the system, or in other words, the "stability" or "strength" it signifies. Actually, this is akin to fan-in or incoming coupling metrics [9]. It is also similar to AIS [Section III]. However, we include fan-out in Dr to factor-in the dependence. The change cost of element ***e***

CC (***e***) = [|IR(***e***)| +1] / [|IR(***e***)| +1+ |OR(***e***)| + |SOR(***e***)|] e≠si ***(PM 9***)

Clearly, IOE (**e**) +CC (**e**) =1.

Similarly, for the special case of *si*

CC (*si*) = |SIR(*s*) | / [|IIR(*s*) | + |SIR(*s*) |] ***e*** = *si* (***PM10***)

Again, IOE (*si*) +CC (*si*) =1

B. Extending McCabe's Cyclomatic Complexity (MCC)

We consider applying McCabe's cyclomatic complexity metric. Perepletchikov et al. [8-10] also caution in their work, and rightly so, that traditional procedural and OO metrics are not immediately applicable to SOS. However, for example, basing on CBO (from CK-metrics-suite), and with some adjustments, they derive some metrics at level of ***e's***, which can be aggregated to higher level of service. However, we do not see any problem in adapting McCabe's cyclomatic complexity (MCC) to SOA, and in fact, we take the cue from Vasconcelos *et al.* [18] who have adapted this to derive a complexity metric for what they call ISA (Information System Architecture). MCC has had wide applicapabilty, having been applied to parallel program [27], concurrent network models [28] and embedded software [25].

(***PM11***): For each operation in a service, there is control flow graph (CFG) across some ***e***'s in a service. This graph is, of course, $cs_1 \cup cs_2 \cup cs_3 \cup \ldots \cup cs_l$, where *cs*'s are *l* number of collaboration sequences in an operation, and ∪ denotes graph union. *cs* is defined in the Perepletichikov model (Section II) as set of elements **e** that are invoked in response to specific inputs on an operation**.** From CFG of each operation, cyclomatic complexity is calculated as usual as (edges-nodes+2). Then, cyclomatic complexities are calculated for all operations in a service. All these values are *added*, giving *cyclomatic complexity for that service*.

C. Inter-connection Complexity

Maertz and Lindner [30] have proposed a complexity metric, what they call complexity of interconnection, for digital instrumentation and control software. However, it is based on generic graph. The metric is basically an average value reflecting the multiplicity of usage of the individual functional blocks for the computation of various output signals of a logic diagram of I&C software. We adapt it as follows. We juxtapose functional block with the service implementation elements (***e***), and output signal to an operation in a service; the entire service being juxtaposed to a logic diagram. Complexity of interconnection V(*s*), where *s* is a service,

V (*s*) =∑ |VE (*o*$_i$) | / |{**e**'s implementing service s}| (***PM12***)

The summation is over all operations o_i.

VE (o_i) denotes the set of ***e***'s implementing o_i. Obviously, each VE(o_i) *may* have some common elements with another VE (o_j). The metric V(*s*) actually gives an average of multiplicity of usage of every element for implementing each operation of the service.

The metric V(*s*) will have a limited range of values, which is in the interval between 1 and the number of operations of a service i.e. 1 ≤V(*s*) ≤ number of operations of service. As such, V(*s*) can be normalized to the interval [0, 1] thus giving a relative metric for the complexity of interconnection.

VIII. MODIFIED RESPONSE FOR OPERATION

The paper [9] does provide a definition for a metric Response for Operation, RFO. RFO for a given operation *o* of a service interface *si* is the cardinality of the *sets* of implementation elements and other service interfaces that can be executed in response to invocations of *o* with all possible parameters. So, RFO = | CS (o)| , as defined in [10], where CS is set of collaboration sequences, *cs*, of operation *o*. The original CK-metrics-suite [29] definition is: The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. This definition has been *strictly f*ollowed in its essence in [18] to derive a metric, Average Response for Service, ARFS, but for Information System Architecture (ISA). We also follow the CK-metrics-definition strictly in its full essence, and derive a metric, Modified Response for Operation, RFO_m.

(***PM 13***): Construct composite graph (which is in fact control flow graph, CFG, in this case) as: $cs_1 \cup cs_2 \cup cs_3 \ldots \cup cs_l$ (graph union over all elements of CS (*o*)). The cardinality of this graph set gives RFO_m (m for modified, to distinguish it from RFO as above).

IX. CONCLUSION AND FUTURE WORK

We have demonstrated in this paper that the existing work on metrics in SOA and related domains of software engineering offer ample scope of expanding and deepening existing models. Thereby, we illustrate that multiple formal models of SOA will benefit the metrics work in SOA.

This paper is intended to be an initial warm-up to our future plan of conducting research along the following tentative plan. We have also proposed MCC for business process workflow, derived the Liu-Traore metric for atomic service, and derived a service granularity metric. We intend to expand our initial work. An important avenue is to investigate all modeling work done so far, especially those that have lead to formulation of important metrics, including both generic models like UML and specific ones, e.g. that of Perepletchikov et al. and conduct a comparison and critical analysis vis-à-vis commonly understood and implemented SOA concepts, and those of software architecture, and in light of current SOA practice. It is likely some consolidated work will reveal more insights and depths, and possibly development of some more quality metrics. Another interesting area could be to explore if and how various metrics are related to other metrics within a suite, and to metrics external to suite. Specifically, metrics from different suites but measuring the same attributes should not reflect any conflict whatsoever.

Recommending thresholds for various SOA metrics is very important from practical-implementation viewpoint. Thresholds [31] are heuristic values used to set ranges of desirable and undesirable metric values for measured software. These thresholds are used to identify *anomalies,* which may or may not be an actual *problem*.

As has been pointed out earlier in this paper, the research in quality, metrics and evaluation for SOA is in early stages. Many of the metrics suites need empirical validation. A metric that is demonstrated to empirically co-relate with the attribute it is supposed to measure finds wide acceptability. This option offers ample avenues for future research.

ACKNOWLEDGMENT

We thank Andre van der Hoek, Mikhail Perepletchikov and Dmytro Rud for providing clarifications and inputs on their work discussed in this paper.

REFERENCES

[1] P. Bianco, R. Kotermanski, and P. Merson, "Evaluating a Service-Oriented Architecture," CMU/ SEI-2007-TR-015. http://www.sei.cmu.edu/pub/documents/07.reports/07tr015.pdf

[2] G. Swart, B. Aziz, S.N. Foley and J. Herbert, "Trading off Security in a Service Oriented Architecture," *In Proc.of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security.* http://www.cs.ucc.ie/~herbert/pubs/ifip2005.pdf

[3] M.P. Papazoglou, and D. Georgakopoulos, "Service-Oriented Computing," *Communications of the ACM*, 2003/ vol.46, No 10

[4] G. Chastek and R. Ferguson, "Toward Measures for Software Architecture," CMU/SEI-2006-TN-013. http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tn013.pdf

[5] B. Portier, "SOA Terminology overview, Part1: Service, architecture, governance, and business terms," 2007. http://www.ibm.com/developerworks/webservices/library/ws-soa-term1/?S_TACT=105AGX04&s

[6] D. Russell and J. Xu, "Service Oriented Architecture in the Provision of Military Capability," *UK e-Science All Hands Meeting*, 2007. http://www.comp.leeds.ac.uk/NEC/doc/SOACapabilityAHM2007.pdf

[7] C. Emig *et al.* , " The SOA's Layers," http://www.cm-tm.uka.de/CM-Web/07.Publikationen/%5BEL+06%5D_The_SOAs_Layers.pdf

[8] M. Perepletchikov, C. Ryan, K.Frampton, and H. Schmidt "Formalising Service-Oriented Design," *Journal of Software*, Vol.3, No. 2, Feb. 2008

[9] M. Perepletchikov, C. Ryan, and K.Frampton, "Coupling Metrics for Predicting Maintainability in Service-Oriented Designs," *In 18th International Conference on Software Engineering (ASWEC2007)*, Melbourne, Australia, 2007

[10] M. Perepletchikov, C. Ryan, and K.Frampton, "Cohesion Metrics for Predicting Maintainability of Service-Oriented Software," *In 7th International Conference on Quality Software*, Portland, USA, 2007

[11] D. Rud, A. Schmietendorf, and R. Dumke, "Product metrics for service-oriented infrastructures," *In Proc. 16th International Workshop on Software Measurement/DASMA Metrik Kongress 2006*, Potsdam, Germany. http://www.cs.uni-magdeburg.de/~rud/papers/Rud-07.pdf
[12] D. Rud, A. Schmietendorf, and R. Dumke, "Resource metrics for service-oriented infrastructures," *In Proc. SEMSOA 2007*, pp. 90-98, May 10-11, 2007, Hannover, Germany . http://www.cs.uni-magdeburg.de/~rud/papers/Rud-13.pdf
[13] T. Xu, K. Qian, and X. He, "Service Oriented Dynamic Decoupling Metrics," *The 2006 Intl. Conf. on Semantic Web and Web Services (SWWS' 06), June 26-29, 2006 WORLDCOMP' 06*, Las Vegas, USA.
[14] Y. Liu and I. Traore, " Complexity Measures for Secure service-Oriented Software Architectures," *In the Proc. of the 3rd IEEE International PROMISE Workshop*, May 20, 2007, Minneapolis, Minnesota, USA
[15] J. Cardoso, "Approaches to Compute Workflow Complexity," *In Dastguhl Seminar Proc. 06291*, http://drops.dagstuhl.de/opus/volltexte/2006/821/pdf/06291.CardosoJorge.Paper.821.pdf
[16] J. Cardoso, "Process Control-flow Complexity Metric: An Empirical Validation," *IEEE Intl. Conf. on Services Computing (IEEE SCC 06)*, Chicago, USA, Sept, 2006, pp 167-173, IEEE Computer Society
[17] W. Zhao, Y. Liu, J. Zhu, and H. Su, "Towards Facilitating Development of SOA Application with Design Metrics," *Service-Oriented Computing - ICSOC 2006, 4th International Conference*, Chicago, IL, USA, December 4-7, 2006.
[18] A. Vasconcelos, P. Sousa and J. Triblolet, "Information System Architectures: An Enterprise Engineering Evaluation Approach," http://www.inesc-id.pt/ficheiros/publicacoes/3543.pdf
[19] N. Looker, J. Xu, and M. Munro, "Determining the Dependability of Service-Oriented Architectures," *International Journal of Simulation and Process Modelling*, pp. 88-97, vol.3, no.1/2,2007
[20] D. Cotroneo, C. Di Flora and S. Russo, "Imroving Dependability of Service Oriented Architectures for Pervasive Computing," *In Proc. of The Eighth IEEE Intl. Workshop on Object-Oriented Real-Time Dependable Systems*
[21] A. Korostelev, J. Lukkien, and J. Nesvadba "Error Detection in Service-Oriented Distributed Systems," *Proc. of IEEE Int. Conf. on DSN 2006*, vol. 2, pp. 278-282, Philadelphia, USA, June 25 - 28, 2006.
[22] V. Gruhn and R. Laue, "Complexity Metrics for Business Process Models," http://ebus.informatik.uni-leipzig.de/~laue/papers/metriken.pdf
[23] A. van der Hoek, E. Dincel, and N. Medvidovic "Using Service Utilization Metrics to Assess the Structure of Product Line Architectures," *Ninth IEEE Software Metrics Symposium*, September 2003.
[24] E. Dincel, N. Medvidovic and A. van der Hoek, "Measuring Product Line Architectures" *Fourth International Workshop on Product Family Engineering*, October 2001, pages 346–352,
[25] G. Menkhaus and B. Andrich "Metric Suite for Directing the Failure Mode Analysis of Emdedded Software Systems," *Proc. Of 7th ICEIS'05* http://www.softwareresearch.net/site/publications/C069.pdf
[26] J. Zhao, "On Assessing the Complexity of Software Architectures," *In Proc. 3rd International Software Architecture Workshop*, pp.163-166, ACM SIGSOFT, ACM Press, November 1998.
[27] S. VanderWiel, D. Nathanson, and D.J. Lilja "Performance and Program Complexity in Contemporary Network-based Parallel Computing Systems," Technical Report No.HPPC-96-02, March 1996, University of Minnesota
[28] N. R. Hall and S. Preiser, "Combined Network Complexity Measures," *IBM J. R. D.*, Vol. 28, No 1., Jan 1984
[29] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Soft. Engg.*, vol 20, no 6, 1994
[30] J. Märtz and A. Lindner, "Complexity Measurement of Software in Digital I&C-Systems for the Quantification of Reliability," *IAEA Technical Meeting (621-12-TM-26932)*, Sept 2005, Chatou, France
[31] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: A Practical Guide*, New Jersey: PTR Prentice Hall, 1994, ch. 1.