Software Metrics for the Efficient Execution of Mobile Services

Pablo Rossi and Zahir Tari

School of Computer Science & IT, RMIT University, Melbourne, Australia
{pablo, zahirt}@cs.rmit.edu.au

Abstract. This paper presents a suite of software code metrics, developed specifically for service-oriented systems with a well-defined methodology, which can be used as indicators of runtime efficiency. Existing literature on software metrics is mainly focused on centralized systems, while work in the area of distributed systems, particularly in service-oriented systems, is scarce. Firstly, a critical analysis of the problem domain identifies a number of software attributes which are likely to have an impact on efficiency. Secondly, concrete metrics are defined and evaluated (theoretically and empirically) for all identified attributes, with results showing that these software metrics are strongly correlated to typical efficiency metrics. Finally, a simple algorithm, which facilitates the runtime adaptation of service-oriented systems via service re-deployment, illustrates a practical application of the metrics.

1. Introduction

Existing literature on software metrics is mainly focused on centralized systems (e.g. [1]), while work in the area of distributed systems, particularly in service-oriented systems, is scarce. Systems with distributed components differ from traditional non-distributed systems along a number of dimensions including communication type, latency, concurrency, partial versus total failure, and referencing/parameter-passing strategies [2]. Distributed systems with service-oriented components are even more complex, since efficiency and other quality attributes must be achieved in a typically more heterogeneous networking and execution environments. Given these differences, this paper argues that it is necessary to extend established software measurement and related techniques before applying them to the emerging domain of service-oriented systems (SOS). Note that it has been argued before, in the domain of web systems, that traditional techniques and metrics should be re-assessed before being applied to a new domain [3].

This work is part of a project whose aim is to design an efficient middleware infrastructure to support highly adaptable mobile services. This infrastructure will provide robust and efficient management of business processes across different enterprises. In this context, adaptation refers to the ability of the software, or the underlying middleware, to modify its behaviour in response to changes in the environment. In this project, adaptation can be achieved, among others, via service mobility, where individual services can migrate through the system nodes. The decision of when and how service migration should be performed is dependent on factors such as available resources and the nature of the interaction between services. As such, this provides a practical application for the metrics proposed in this paper.

This paper presents a suite of software code metrics that can be used as indicators of runtime efficiency of service-oriented systems. These metrics were developed taking into account the particular characteristics of service–oriented systems (SOS) and following a well-defined methodology. The availability of well-defined metrics is crucial for middleware infrastructures to make dynamic decisions at run-time that are objective and robust.

The rest of this paper is organized as follows: Section 2 begins with a review of related work, with an emphasis on prior studies involving the specification of metrics for distributed systems. Section 3, through a critical analysis of the problem domain which provides face validity [4], identifies a number of specific software attributes that are likely to have an impact on efficiency, with a concrete metric defined for each. Since theoretical validation of software measures provides supporting evidence as to whether a measure really captures the internal attributes they purport to measure, metrics are theoretically validated in section 4. Section 5 evaluates empirically the relationships between software and efficiency metrics in the context of SOS. To illustrate the potential practical applications of the metrics, a simple strategy is presented in section 6, with the intention of facilitating runtime decisions concerning the adaptation of SOS via service mobility. Finally, section 7 closes with a summary, conclusions and discussion of future work.

2. Related Work

Although many software metrics have been defined for traditional systems [5], a much smaller number relate to distributed systems in general, and few, if any, consider the unique characteristics of SOS as is the subject of this paper. This section provides a review of related work in terms of the measurement of software attributes of distributed systems.

Shatz [6] proposed a metric for measuring communication complexity in distributed Ada programs, describing total complexity as the weighted sum of two components. Firstly, local complexity, which reflects the complexity of the individual tasks (disregarding their interactions with other tasks), was measured using traditional metrics such as lines of code or cyclomatic complexity. Secondly, communication complexity, which reflects the complexity of interactions among tasks, was derived by representing the programs as Petri nets and measuring the number of rendezvous which can be executed concurrently at a given point in time. Neither theoretical/empirical evaluation nor discussion about the practical utility was presented.

Cheng [7] also proposed a set of complexity metrics for distributed programs. Like Shatz [6], the metrics were defined based on graph representations for 1) multiple control flows (non-deterministic parallel control-flow net), 2) multiple data flows (non-deterministic parallel definition-use net) and 3) various program dependencies (process dependency net). As above, no empirical support was provided, nor was a discussion of how the metrics could be used within the software engineering process.

Based on the smallest event communication group (SECG) concept, Tsuar and Horng [8] suggested a metric to quantify the complexity of distributed programs, which was defined in terms of the number of events of a SECG. Although, the metric was experimentally evaluated with a moderately complex example, they were not evaluated theoretically, and it was not made clear how it could be used by practitioners.

Morasca [9] put forward a set of measures for capturing a number of internal attributes (namely size, length, complexity, and coupling) of software specifications written with Petri nets for concurrent systems. These measures were theoretically evaluated, but empirical evaluation was not provided.

A measurement suite to quantify design attributes of distributed systems was presented by Rossi and Fernandez [10]. The proposed measures were obtained from formal models derived from an analysis of the problem domain. Although these measures were theoretically evaluated, only a small subset was subject to empirical evaluation [11].

Arguably the closest study to our work are the metrics proposed by Ryan and Rossi [12], since they were defined and empirically evaluated for distributed systems with mobile components. However, this work focuses on the specific characteristics of software objects and, as such, may not be directly applicable to services.

In summary, existing studies suffer at least one of the following shortcomings:

- metrics are not theoretically or empirically evaluated
- metrics have no clear practical applicability
- metrics do not capture the particular nature of SOS

Therefore, given these shortcomings, it seems appropriate to develop and evaluate a new suite of software metrics to support the unique aspects of SOS.

3. Analysis of the Problem Domain

Here we are concerned with the impact of software attributes on the efficient execution of services in a mobile computing context. For the purpose of this paper, in line with ISO 9126-1 [13], *efficiency* is considered to be a high-level quality attribute comprising the attributes *performance* (or time behaviour) and *resource utilization*. Software is considered to be more efficient as performance increases and resource utilisation decreases. These attributes are quite broad and, as such, were decomposed into particular sub-attributes of interest:

P1. *Service Migration Cost*: the cost of moving a service between hosts.
P2. *Operation Execution Cost*: the processing cost of an operation, ignoring any overhead associated with its call.
P3. *Operation Call Cost*: the cost of calling a service operation, independent of its actual execution.

R1. *Memory Utilisation*: the current memory usage on a host.

R2. *Network Utilisation*:the current unavailable network bandwidth of a host
R3. *Processor Utilisation*: the current processing load of a host

It should be noted that other resources such as mass-storage capacity and power consumption are also considered important but due to space constraints will be studied in future work.

From a critical analysis of the problem domain a number of software attributes were identified that were likely to have an impact on the efficiency of SOS in a mobile environment. In this context, the *size* and *coupling* of a service were identified as the key software attributes that represent most of the impact on efficiency. As size and coupling are also generic attributes, they were refined to capture more precisely the specific characteristics of services as software components:

S1. *Service Implementation Dimension*: the size of the service executable code; the larger the Service Implementation Dimension, the longer the *Service Migration Time* and the higher the *Network Utilisation*.
S2. *Service State Dimension*: the size of a service execution state; the larger the Service State Dimension, the higher the *Memory Utilisation*.
S3. *Operation Interface Dimension*: the aggregated size of the parameters of a service operation interface; the larger the Operation Interface Dimension, the higher the *Operation Call Cost* and the higher the *Network Utilisation*.
S4. *Operation Execution Length:* the length of a service operation implementation; the larger the Operation Execution Length, the higher the *Operation Execution Cost* and the higher the *Processor Utilisation*.

C1. *Service Collaboration Coupling*: a service degree of connection to other services; the higher the Service Collaboration Coupling, the higher the *Operation Execution Cost*, and the higher the *Network Utilisation*.
C2. *Operation Call Occurrence*: the call frequency of a service operation; the higher the Operation Call Occurrence, the higher the *Network Utilisation* and *Processor Utilisation*.

The model depicted in Figure 1 summarises the studied relationships between software and efficiency attributes — additional attributes such as probability of service execution were identified but considered beyond the scope of this paper. Furthermore, the authors identified other potential relationships that are not expressed explicitly. For example as memory utilisation increases, paging could affect performance; as processor utilisation increases there will inevitably be an effect on attributes such as Operation Call Cost. However since these factors were not expected to have a primary effect, and in the interests of studying a manageable set of metrics in this paper, the analysis of such attributes and relationships is left to future work.

The final stage of the analysis process was to derive concrete metrics for each of the attributes in a form that could be measured at runtime within a middleware infrastructure. The complete set of metrics and their units of measurement are listed in Table 1. A formal definition of the metrics is presented in the next section, while details of how each metrics can be measured in practice can be found in the appendix.

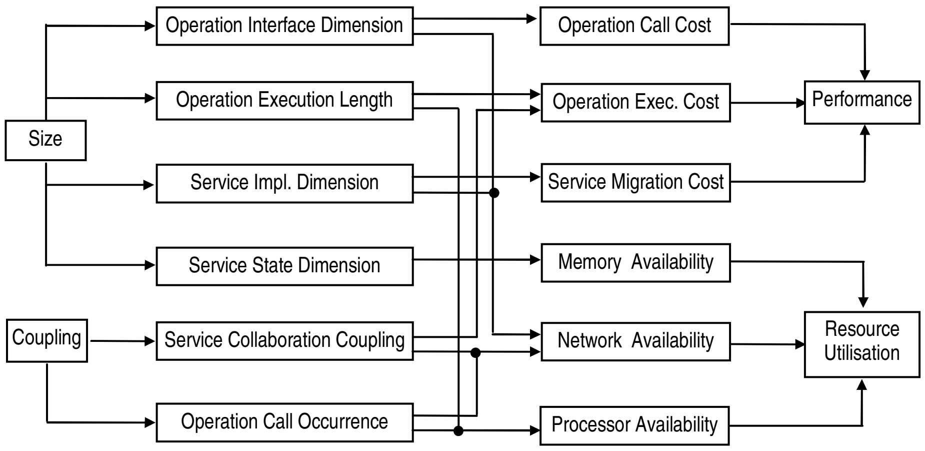

Fig. 1. Summary of the relationships between software and efficiency attributes for SOS

Table 1. Attributes of interest and associated metrics

Attribute		*Metric*	*Unit*
S	Operation Interface Dimension	Operation Interface Size (OIS)	byte
	Service Implementation Dimension	Service Code Size (SCS)	byte
	Service State Dimension	Service State Size (SSS)	byte
	Operation Execution Length	Operation Number of Statements (ONS)	int
C	Operation Call Occurrence	Operation Call Number (OCN)	int
	Service Collaboration Coupling	Collaborator Service Number (CSN)	Int
P	Operation Execution Cost	Operation Execution Time (OET)	ms
	Operation Call Cost	Operation Call Time (OCT)	ms
	Service Migration Cost	Service Migration Time (SMT)	ms
RU	Memory Utilisation	Memory Availability (MA)	byte
	Network Utilisation	Network Availability (NA)	byte/s
	Processor Utilisation	Processor Availability (PA)	int/s

4. Theoretical Evaluation

Since theoretical validation of software measures provides supporting evidence as to whether a measure really captures the internal attributes they purport to measure, we consider this validation as a necessary step before empirical validation takes place. The distance framework [14] is briefly introduced in sub-section 4.1 and it is then used to define the proposed metrics formally in the following sub-sections. This framework has been employed to theoretically validate software measures previously (e.g. [15], [16]).

4.1. Distance Framework

The distance-based approach presents a set of measure axioms whose sufficiency is assured by measurement theory, and a constructive procedure that defines software measures satisfying these axioms. These axioms are the metric axioms, used in Mathematics to define measures, an extension of the notion of distance. This section summarizes the basic concepts used here to make the paper self-contained. (For more details refer to the original work [14]). The distance-based measure construction process consists of five steps:

1. For the set of software entities E and for the internal attribute a, select a set of software entities M that can be used as measurement abstractions to emphasise a, and define a function $\alpha: E \rightarrow M$.
2. Define a set T of elementary transformation types on M that is constructively and inverse constructively complete to model the conceptual distances between measurement abstractions.
3. Quantify distances between measurement abstractions defining a metric $\delta: M \times M \rightarrow \Re$ such that (M, δ) is a metric space.
4. Select a reference model $\tau \in M$ that is the software entity abstraction for which it holds that for all $e \in E$ with $\alpha(e) = \tau$, e has the lowest value of a.
5. Define a function $\mu: E \rightarrow \Re$ such that for all $e \in E$, $\mu(e) = \delta(\alpha(e), \tau)$ which is a measure of distance from $\alpha(e)$ to τ.

4.2. Coupling Metrics

Here we provide the formal definition of CSN that demonstrates its theoretical validity. OCN was formally defined and validated following an analogous process

Step 1: The set of software entities E is the universe of services (US) that is relevant for some system domain and S is a service (viz. $S \in US$). The attribute of interest a is the number of services that collaborate with S via operation calls. The set of services that collaborate with service S is then a subset of US. All the sets of services with collaboration coupling within US are elements of the power set of US, denoted by $P(US)$. Consequently we can associate the set of measurement abstractions M to $P(US)$ and define the abstraction function $\alpha_{CSN}: US \rightarrow P(US)$ as:

$$\forall S \in US: \alpha_{CSN}(S) = \{R \in US \mid R \text{ collaborates with } S \text{ via an operation call}\} \quad \textbf{(1)}$$

This function maps a service S onto the set of collaborator services that are called by S.

Step 2: The next step is to model the distance between elements of M. It is necessary to find a set of elementary transformation types for $\mathbf{P}(US)$ such that any set of services can be transformed into any other set of services by way of a finite sequence of transformations. Since the elements of $\mathbf{P}(US)$ are sets of components, T must only contain two types of elementary transformation. $T = \{\theta_{CSN1}, \theta_{CSN2}\}$ where

$$\forall s \in \mathbf{P}(US): \theta_{CSN1}(s) = s \cup \{m\}, \text{ with } m \in US, \qquad \textbf{(2)}$$
$$\forall s \in \mathbf{P}(US): \theta_{CSN2}(s) = s - \{m\}, \text{ with } m \in US.$$

Given two sets *s* and *s'* of **P**(*US*), *s* can always be transformed into *s*' by first removing all services from *s* that are not in *s'* (through θ_{CSN2}) and then adding all services to *s* that are in *s*' – *s* (through θ_{CSN1}).

Step 3: The distance between two sets of services *s* and *s'* can be measured by the length of the shortest sequence of elementary transformations taking *s* to *s'*. As exactly one elementary transformation will be needed for each service of *US* that is contained in either *s* or *s'*, but not in both sets, the distance value is equal to the cardinality of the symmetric difference between *s* and *s'*:

$$\forall s, s' \in \mathbf{P}(US): \delta_{CSN}(s, s') = |s - s'| + |s' - s| \qquad \textbf{(3)}$$

Step 4: The reference abstraction is the empty set of services. It is desirable that a service *S* without service collaborations will have the lowest possible value for the *CSN* measurement. Hence we define the following function: τ_{CSN}: $US \rightarrow \mathrm{P}(US)$: $S \rightarrow \emptyset$.

Step 5: The number of services called by service $S \in US$, can be formally defined as the distance between its set of collaborator services and the empty set. Therefore, formally *CSN* can be defined as μ_{CSN}: $\mathrm{P}(US) \rightarrow \Re$:

$$\forall S \in US: \mu_{CSN}(S) = \delta_{CSN}(\alpha_{CSN}(S), \tau_{CSN}) = |\alpha_{CSN}(S)\Delta\emptyset| = |\alpha_{CSN}(S)| \qquad \textbf{(4)}$$

4.3. Size Metrics

Here we theoretically validate ONS by presenting its formal definition. SCS, SSS and OIS were formally defined and validated following an analogous process.

Step 1: The set of software entities *E* is the universe of Operations (*UO*) that is relevant for some Service domain and *O* is an Operation (viz. $O \in UO$). Let *UES* be the Universe of Executable Statements relevant to *O*. The attribute of interest *a* is the number of Executable Statements that are part of Operation *O*. The set of Executable Statements *ES* that are part of Operation *O* is then a subset of *UES*. All the sets of Executable Statements that are part of Operations within *UO* are elements of the power set of *UES*, denoted by **P**(*UES*). Consequently we can associate the set of measurement abstractions *M* to **P**(*UES*) and define the abstraction function α_{ONS}: $UO \rightarrow \mathbf{P}(UES)$ as:

$$\forall O \in UO: \alpha_{ONS}(O) = \{ES \in UES \mid ES \text{ is part of } O\} \qquad \textbf{(5)}$$

This function maps an Operation *O* onto its set of Executable Statements.

Step 2: The next step is to model the distance between elements of M. It is necessary to find a set of elementary transformation types for $\mathbf{P}(UES)$ such that any set of Executable Statements can be transformed into any other set of Executable Statements by way of a finite sequence of transformations. Since the elements of $\mathbf{P}(UES)$ are sets of Executable Statements, T must only contain two types of elementary transformation. $T = \{\theta_{ONS1}, \theta_{ONS2}\}$ where

$$\forall es \in \mathbf{P}(UES): \theta_{ONS1}(es) = es \cup \{m\}, \text{ with } m \in UES, \qquad \textbf{(6)}$$
$$\forall es \in \mathbf{P}(UES): \theta_{ONS2}(es) = es - \{m\}, \text{ with } m \in UES.$$

Given two sets es and es' of $\mathbf{P}(UES)$, es can always be transformed into es' by first removing all Executable Statements from es that are not in es' (through θ_{ONS2}) and then adding all Executable Statements to es that are in $es' - es$ (through θ_{ONS1}).

Step 3: The distance between two sets of Executable Statements es and es' can be measured by the length of the shortest sequence of elementary transformations taking es to es'. As exactly one elementary transformation will be needed for each statement of UES that is contained in either es or es', but not in both sets, the distance value is equal to the cardinality of the symmetric difference between es and es':

$$\forall es, es' \in \mathbf{P}(UES): \delta_{\text{ONS}}(es, es') = |es \,\Delta\, es'| \qquad \textbf{(7)}$$

Step 4: The empty set of statements is the reference abstraction τ. It is desirable that an Operation O without Executables Statements will have the lowest possible value for the ONS measurement. Hence we define the following function: τ_{ONS}: $UO \rightarrow \mathbf{P}(UES)$: $O \rightarrow \varnothing$.

Step 5: The number of Executable Statements that are part of Operation $O \in UO$, can be formally defined as the distance between its set of Executable Statements and the empty set. Therefore, formally ONS can be defined as μ_{ONS}: $\mathbf{P}(UO) \rightarrow \Re$:

$$\forall O \in UO: \mu_{ONS}(O) = \delta_{ONS}(\alpha_{ONS}(O), \tau_{ONS}) = \qquad \textbf{(8)}$$
$$|\alpha_{ONS}(O) - \varnothing| + |\varnothing - \alpha_{ONS}(O)| = |\alpha_{ONS}(O)|$$

5. Empirical Evaluation

This section provides experimental results to support the relationships between software and efficiency metrics described in section 3. We have followed some of the guidelines provided in the literature [17] on how to perform and report controlled experiments. (Please note that not all available information has been included in this paper due to space constraints.)

5.1. Definition

Following the GQM template [18], our experiment goal can be summarised as follows:

Analyse SOS software measures,
for the purpose of evaluating,
with respect to their capability of being used as indicators of runtime efficiency,
from the point of view of SOS engineers,
in the context of wireless and mobile environments.

5.2. Planning

In order to evaluate software measurement hypothesis empirically, it is possible to adopt two main strategies [19]: (a) small-scale controlled experiments, and/or (b) real-scale industrial case studies. In this case we chose the first alternative, since it is more suitable to study the phenomena of interest in isolation, without having to deal with other sources of variation, such as co-existing systems, security mechanisms, etc. However, we envisage that after several experiments the suite of measures will be shown to be robust, and we intend to test the measures following the second strategy.

The hypotheses to be tested were derived from the attribute relationships discussed in section 3 as part of the analysis of the problem domain. The dependent (efficiency) and independent (software) variables are quantified by the metrics shown in Table 1. Details of how each measure was quantified can be found in the Appendix.

For each hypothesis, experimental data was collected using a synthetic Java system, with the measurement of metrics obtained either through internal instrumentation of the code, or from the operating system via a native interface. All tests were executed in an isolated network using two identical laptops in a client server configuration via wireless link.

5.3. Operation

Before the actual experiment, several pilot experiments were run to make sure that there were no apparent anomalies, and the system behaved in the same way as before the measurement code was introduced.

The experiment was conducted in the Distributed Computing Research Laboratory of our University. The system was executed on an isolated (54 Mbps) wireless network of laptops (Pentium M 1.6, GHz, 512MB RAM) and running under a Windows operating system. All computers had the same hardware and software, and were configured in the same way. Every service was run on a separate laptop as the only user process, all other processes running were a few system processes started by default. The execution of the system was initiated and terminated by the experiment team, which also controlled that in the meantime nobody else had access to the facilities.

Despite the data being collected reliably and objectively by electronic means, it was thoroughly inspected to assert that it was consistent. For this purpose we run the experiment three different times and compared the three data sets obtained. However, it should be noted that only the first data set was subject to analysis. Finally, there was no need to discard any data, hence all data collected was used.

5.4. Analysis and Interpretation of the Results

After the execution of the experiment, all the measures were computed electronically from the recorded data. The empirical data was analysed with the assistance of the statistical software package SPSS [20], and the obtained results are presented in the remainder of this section.

Correlation Analysis. Table 2 presents the Pearson correlation coefficients (significant at the 0.01 level) between the software measures and efficiency measures.

Table 2. Pearson correlation coefficients (N = 100)

	OET	OCT	SMT	MA	PA	NA
SCS			0.927			0.928
SSS				0.959		
CSN	0.932					0.938
OIS		0.955				0.965
ONS	0.990				0.967	
OCN					0.931	0.984

Discussion. The results show that all the associations are statistically significant. The correlation coefficients are significant, indicating a nontrivial association of the software measures with the efficiency measures. This suggests that these variables are candidates for a base regression model to estimate efficiency. Examination of the coefficients indicates that all software measures are positively correlated to the efficiency measures — it should be noted that a higher value of OET, OCT or SMT indicates worse performance.

Univariate Regression Analysis. Here we present the results obtained when analysing the individual impact of the software measures on efficiency using Ordinary Least Squares (OLS) Regression [21]. In general, a multivariate linear regression equation has the following form:

$$Y = B_0 + B_1X_1 + \ldots + B_nX_n \qquad (\mathbf{9})$$

where Y is the response variable, and X_i are the explanatory variables. A univariate regression model is a special case of this, where only one explanatory variable appears. Table 3 presents the unstandardised regression coefficients (B_i), the statistical significance of B_i (p_i), and the goodness-of-fit (R^2) of models. Each row contains the statistics of a different univariate regression model.

Discussion. The results obtained are remarkably consistent. They indicate that all software measures that we considered in this paper indeed strongly correlate with efficiency. In the best case, in our context, ONS accounted for 98 percent of the variation in performance (measured by OET)—each increase of one unit of ONS increased OET by 16.88 units. In addition, by analysing the trends indicated by the coefficients, we see that the hypotheses underlying the measures are empirically supported.

It should be noted the fact that $p_0 > 0.01$ for some models only means that we cannot really conclude that $B_0 \neq 0$, but this does not affect the fact that we can realistically conclude that it is very unlikely that $B_1 \neq 0$, i.e., it is very likely the software attribute is correlated to the efficiency attribute.

Table 3. Univariate Regression Models

X	*Y*	B_0	B_1	p_0	p_1	R^2	*N*
SCS	SMT	22983	3.701	0.000	0.000	0.859	100
SCS	NA	74065	2.809	0.000	0.000	0.860	100
SSS	MA	0.000	1.044	0.000	0.000	0.920	100
CSN	OET	7.214	2.345	0.206	0.000	0.868	100
CSN	NA	32516	5311.7	0.159	0.000	0.879	100
OIS	OCT	5059	0.436	0.000	0.000	0.912	100
OIS	NA	-2463.3	2.839	0.764	0.000	0.931	100
ONS	OET	-44002	16.884	0.081	0.000	0.981	100
ONS	PA	10920	65.249	0.530	0.000	0.936	100
OCN	NA	-288.2	168.239	0.462	0.000	0.968	100
OCN	PA	342.1	16.042	0.000	0.000	0.867	100

Validity. Four different threats to the validity of the study were addressed [17]:

- *Conclusion validity*. An issue that could affect the statistical validity of this study is the size of the sample data, which may not be large enough for a conclusive statistical analysis. We are aware of this, so we do not consider these results to be final.
- *Construct validity*. The study was carefully designed, and the design was piloted several times before actually being run. The efficiency metrics are obtained from the OS, thus it is assumed they are reliable. The software metrics used in this study were shown to adequately quantify the attribute they purport to measure in section 4.
- *Internal validity*. The study was highly controlled and monitored, so it is very unlikely that undetected influences have occurred without our knowledge. The instrumentation was trustworthy since the data was collected, and the measures computed, electronically.

– *External validity*. Although the study is based on a representative case, more studies are needed using real systems. We are also aware that more experiments with different platforms (e.g. computer and network hardware, operating systems, etc.) and infrastructure (e.g. middleware type, programming language) must also be carried out to further generalize these results.

6. Practical Applicability

Software measurement is not merely about defining new metrics, but about building new theories that can help solve practical problems [22]. As stated previously, one of the main goals of this research is to provide middleware support enabling SOS to maintain specified levels of quality, particularly efficiency, in mobile environments.

Consequently, this section defines, and provides initial testing for, an adaptation approach based on the metrics introduced in section 3. In general, the practical application of these metrics is within an middleware infrastructure, which will collect software metrics describing the services that constitute the system, as well as information about the hosts in which the services are running, and will make decisions in terms of service (re) location.

The approach to adaptation developed is decentralized and reactive, involving an individual node making a decision to move one or more services to another host when either a performance or resource utilisation threshold is met. Other approaches to adaptation are possible but are outside the scope of this paper — for example an alternative approach to adaptation could be centralised and proactive involving the solution of an optimization model.

A reactive/adaptive decision is typically triggered by the utilisation of a resource that at some point in time exceeds a predetermined threshold. The objective of this adaptation approach is to distribute the utilisation of resources whilst maintaining (or improving) performance. However, performance and resource utilisation are attributes that generally conflict with each other and since compromises may have to be reached; this decision is not trivial even for the simplest scenario of two services and two machines. A more typical case may involve numerous services and many nodes, and thus achieving an effective decision requires an appropriate process.

Therefore, in order to test the ability of the metrics to support such decision making, whilst yielding a tangible benefit in terms of efficiency, a preliminary empirical study was conducted by implementing a prototype SOS which consists of five main services executing over Sun System Application Server Platform Edition 9. For the experiment, software and performance metrics were collected directly via instrumentation in the system code and resource utilisation metrics from the Windows Performance Monitor via the Java Native Interface (JNI). The experiment was conducted under the same laboratory conditions, using three nodes of the same specification as described in the previous section.

The adaptation decisions (which determine if and when a given service should migrate to another host) were calculated and executed offline. Therefore, the intention of this experiment is not to evaluate the efficiency of the metric collection and adaptation process itself, although this is the subject of ongoing work. Rather, this experiment aims to demonstrate that the metrics presented herein can support the effective placement of services to hosts in a SOS, in order to improve efficiency compared with the baseline case of no adaptation.

Table 4. High-level Adaptation Algorithm

```
maxIndicator = 0, maxService = null, maxNode = null
for each service s in local node do
  for each remote node n do
    i = evaluate(s, n)
    if (i > maxIndicator) then
      maxIndicator = i
      maxService = s
      maxNode = n
    end if
  end for
end for
if (maxIndicator > 0.5) move maxService to maxNode
```

At the abstract level, the adaptation approach operates according to algorithm depicted in Table 4 in which individual nodes move services to other hosts when some criteria related to efficiency (performance versus resource utilisation) are met. The algorithm evaluates, based on the metric values, possible migration options based on the available local mobile services and remote nodes. The algorithm stops when all possible migrations have been evaluated. An explanation of how the function 'evaluate' produces its indicators is given in Table 5.

In the initial state all services (i.e. S_1, S_2, S_3, S_4 and S_5) were residing on node 1 (N_1), where the processor was heavily loaded (PA around 10%), while N_2 and N_3 were not loaded at all (PA around 99%). Applying the adaptation algorithm to SOS resulted in four cases of service migration: S_2, S_3, S_4 and S_5 from N_1 to N_2. The standard deviation of the processor loads was calculated before and after adaptation. Additionally, response time data was collected for all business processes with each executed and measured 100 times.

The standard deviation of the processor loads before adaptation was around 57% whereas the standard deviation after adaptation was around 48%. Moreover, the average process response time after adaptation was 297 ms versus 321 ms before adaptation. Therefore not only did the adaptation algorithm provide better processor load balance, which was the principal aim of this experiment, but it also provided superior performance and thus greater efficiency as well. The adaptation algorithm was initialised with the following parameters: $W_P = 0$, $W_{RU} = 1$, $k_{RU} = 1$, $max_{RU} = 100$.

Although the presented adaptation algorithm is relatively simple, it illustrates the benefits of applying the metrics proposed in this paper to a practical application. This provides a basis for further large scale studies and the development of advanced adaptation approaches. For example, a more sophisticated approach to adaptation based on these metrics and a comprehensive evaluation, can be found in a separate study [23].

Table 5. Function Evaluate Description

There are a number of possible general approaches to decision making based on multiple attributes, which differ in terms of how they specify criteria for the decision making process. Since this is a proof-of-concept implementation of adaptation, we have selected a simple linear additive approach according to the following equation:

$$I_E = (W_{RU}\, I_{RU} + W_P\, I_P) \qquad (\mathbf{10})$$

In order to evaluate such function, and thus produce an overall decision making indicator of efficiency (I_E) that can be used to rank and runtime actions, the level of satisfaction of the individual indicators (I_i) must be calculated—this is done by normalising the values to the unitary interval ($0 \leq I_i \leq 1$). Furthermore, the aggregate decision making function include weights (W_i), to represent the relative importance of the individual indicators when calculating the decision making indicator I_E. A further requirement of the function is that ($W_1 + W_2 + \ldots + W_m$) = 1, where $W_i \geq 0$ for $i = 1 \ldots m$.

Finally, the resource utilisation (I_{RU}) and performance (I_P) indicators were calculated as follows:

$$I_P = \frac{0.5 + 0.5 \times (dif_P - k_P)}{2 \times max_P} \qquad (\mathbf{11})$$

$$I_{RU} = \frac{0.5 + 0.5 \times (dif_{RU} - k_{RU})}{2 \times max_{RU}} \qquad (\mathbf{12})$$

$$dif_P = \left[\sum_{i=1..O_S} \left(\mathbf{OCN}_i \times (ort_i^C - ort_i^D) \right) \right] - \mathbf{SMT} \qquad (\mathbf{13})$$

$$dif_{RU} = \left| \frac{ra^C}{rc^C} - \frac{ra^D}{rc^D} \right| - \left| \frac{ra^C + ru_S}{rc^C} - \frac{ra^D - ru_S}{rc^D} \right| \qquad (\mathbf{14})$$

where:

- max_P and m_{RU} are maximum values of performance and resource utilisation.
- k_P and k_{RU} are threshold parameters which specify the minimum acceptable value of performance and resource utilisation.
- O_S = number of operations of service S.
- ort_i = **OET** + **OCT** (operation response time)
- ru_S = the resource usage of service S: nu_S = **SCS**, mu_S = **SSS**, or pu_S = avg(**ONS**)
- ra^C and ra^D = resource (i.e. network, memory or processor) availability of the current and destination nodes of the service; ra = **NA**, **MA** or **PA**.
- rc^C and rc^D = resource (i.e. network, memory or processor) capacity of the current and destination nodes of the service

7. Concluding Remarks

Having recognized that distributed systems differ from their traditional centralised counterparts, and that SOS are even more complex, this paper has introduced a suite of metrics for such systems. These metrics aim at estimating the impact of software attributes upon efficiency, in terms of performance and resource utilisation, for SOS operating in mobile environments.

Having defined such metrics from a critical analysis of the problem domain, a series of hypotheses relating the metrics were proposed, and the metrics were evaluated theoretically and empirically. With the results demonstrating the strong correlation between software attributes and efficiency of SOS, an adaptation strategy was developed, in order to illustrate one of the practical applications of the metrics in the context of middleware infrastructures for SOS.

Although this paper has made a significant incursion into an area that is not yet well understood, there are a number of limitations and opportunities that remain to be explored in the future which include, but are not limited to:

- Evaluation of the software attributes and their associated metrics against other quality attributes such as reliability.
- Analysis of additional resources such as mass-storage and power
- Evaluation of the overhead of the metric collection and the adaptation strategy.

In closing, the present authors believe this paper to be one of the few reported studies of metrics for distributed software, and the first involving the specific case of SOS. The results encourage further large-scale studies and which will suggest modifications to the metrics suite as additional understanding is achieved.

Acknowledgments. We would like to thank the Australian Research Council (ARC) for supporting this project. This work is funded under ARC Linkage Project scheme no. LP0455234.

Appendix. The contents of this appendix can be obtained from the authors on request or can be found online at http://goanna.cs.rmit.edu.au/~pablo/wewst/appendix.pdf.

References

1. Purao, S. and V. Vaishnavi, *Product metrics for object-oriented system.* ACM Computing Surveys, 2003. **35**(2): p. 191-221.
2. Emmerich, W., *Engineering Distributed Objects*. 2000: Wiley.
3. Ruhe, M., R. Jeffery, and I. Wieczorek. *Using Web objects for estimating software development effort for Web applications*. in *Ninth International Software Metrics Symposium*. 2003.
4. Henderson-Sellers, B., *Object-Oriented Metrics: Measures of Complexity*. 1996, Upper Sadle River, USA: Prentice Hall.
5. Fenton, N. and S. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. Second ed. 1996, London: International Thompson Computer Press.

6. Shatz, S., *Towards Complexity Metrics for Ada Tasking.* IEEE Transactions Software Engineering, 1988. **14**(8): p. 1122-1127.
7. Cheng, J. *Complexity metrics for distributed programs.* in *International Symposium on Software Reliability Engineering*. 1993: IEEE.
8. Tsuar, W. and S. Horng, *A New Generalised Software Complexity Metric for Distributed Programs.* Information and Software Technology, 1998. **40**(5-6): p. 259-269.
9. Morasca, S. *Measuring attributes of concurrent software specifications in Petri nets.* in *Sixth International Software Metrics Symposium*. 1999.
10. Rossi, P. and G. Fernandez. *Definition and validation of design metrics for distributed applications.* in *Ninth International Software Metrics Symposium*. 2003. Sydney: IEEE.
11. Rossi, P. and G. Fernandez. *Design Measures for Distributed Information Systems: an Empirical Evaluation.* in *International Workshop on Software Audit and Metrics (In conjunction with ICEIS)*. 2004. Porto.
12. Ryan, C. and P. Rossi. *Software, Performance and Resource Utilisation Metrics for Context Aware Mobile Applications.* in *Proceedings of International Software Metrics Symposium IEEE Metrics 2005*. 2005. Como, Italy.
13. ISO/IEC, *Information Technology - Software Product Quality - Part 1: Quality Model*. 2003, International Standards Organisation: Geneva.
14. Poels, G. and G. Dedene, *Distance-based software measurement: necessary and sufficient properties for software measures".* Information and Software Technology, 2000. **42**(1).
15. S. Abrahao, et al. *Defining and Validating Metrics for Navigational Models.* in *Ninth International Software Metrics Symposium*. 2003: IEEE.
16. Marcela, G., M. David, and P. Mario, *Defining Metrics for UML Statechart Diagrams in a Methodological Way*, in *Conceptual Modeling for Novel Application Domains (LNCS 2814)*. 2003, Springer. p. 118-128.
17. Wohlin, C., et al., *Experimentation in Software Engineering*. 2000: Kluwer.
18. Basili, V. and D. Rombach, *The TAME Project: towards improvement-oriented software environments.* IEEE Transactions Software Engineering, 1988. **16**(6).
19. Briand, L., S. Morasca, and K. El Emam, *Theoretical and Empirical Validation of Software Product Measures*. 1995, International Software Engineering Research Network.
20. SPSS, I., *SPSS 8.0: User Guide*. 1998, Chicago: SPSS Inc.
21. Freund, R. and W. Wilson, *Regression Analysis: Statistical Modeling of a Response Variable*. 1998: Academic Press.
22. Briand, L.C., S. Morasca, and V.R. Basili, *An operational process for goal-driven definition of measures.* Software Engineering, IEEE Transactions on, 2002. **28**(12): p. 1106-1125.
23. Rossi, P. and Z. Tari. *Software Adaptation for Service-Oriented Systems.* in *Middleware for Service Oriented Computing (MW4SOC'06)*. 2006. Melbourne, Australia: ACM Press.