Function Points and Service-Oriented Architectures

A reference model for component based architectures

Roberto Meli
Data Processing Organization Srl
Rome, Italy
email: roberto.meli@dpo.it

***Abstract* — Software Service Oriented Architectures (SOA) are characterized by the distribution of data processing components on separate and cooperating technology platforms. Unfortunately, this model represents a technology perspective that is not useful to identify software objects to be measured starting from the user's (business) point of view, as required by international standards of functional measurement (ISO 14143). To solve this problem we have defined the concept of Measurable Software Application (MSA). In the proposed model, each MSA must lie in one and only one layer but may use or incorporate services belonging to different layers. In each layer, we can find generalized software components designed to give a specific and reusable support to the implementation of particular functional or non-functional requirements of the business (application) layer. The identification of generalized software components that belong to lower level layers with respect to the business one, is also crucial to quantify the rate of reuse that should be computed in each project measure for the correct calculation of economical reward, as defined in a customer-supplier contract.**

Keywords-function point analysis; SOA; component; middleware; measurement.

I. INTRODUCTION

The goal of this paper is to show a model of a software application specifically suited to allow Functional Size Measurement Methods (also known as Function Points) to be applied to this kind of architecture.

Software Service Oriented Architectures (SOA) are characterized by the distribution of data and processing logic components among separate and cooperating technology platforms [1]-[5].

The execution of a process is often implemented dynamically on the most appropriate element of the architecture at any given time. This organization allows to reuse generalized components (often called services), through standardization and specialization, in order to construct any new transaction. Models that describe these architectures use the concept of layers, which is a way to aggregate those components on the basis of homogeneity of logical representation and usage.

These model, however, are seen from a technological perspective, oriented to the software design and implementation rather than the to the final user point of view.

One of the main values of Function Point Analysis is to allow comparing the convenience of implementing the same user functional requirements (same FP size) with competitive architectures in such a way to choose the most productive one. If any different technical organization of the implemented user requirements had a different logical size this comparison would not be possible. On the other side we should have a practical way to allocate size and consequently effort and costs in the places where that effort and cost is originated. This is the goal that we had in mind in proposing the following approach.

Section II recalls some concepts related to a typical SOA architecture; Section III presents a model for functional measurement of component based architecture; Section IV illustrates how to consider embedded services provided by an application to another one; Section V presents a comparison with related works; Section VI explains the needs for further research and finally Section VII shows the conclusions.

II. A TYPICAL SOA MODEL

In a distributed architecture, the business layer is associated to the user needs and the typical way of using a system requested by the final user. A Data Base Management System (DBMS) layer, instead, is associated to the requirements of data treatment and storage regardless to their semantic content for the end user; in other words, it is a layer that manipulates the information from a structural point of view rather than from the final user semantic point of view. To the DBMS point of view, the user "meaning" of a table and its data fields is not important. The structured relationships between tables and data fields, the integrity rules, the allowed operations, etc., are important independently by the business meaning of the entity represented by the table(s).

The most used layers to aggregate software components are (Figure 1):

- Presentation Layer: contains the user interface, typically the internet browser. From this layer it is only possible to call services/classes that are in the Business Layer.
- Business Layer: contains services/classes that perform the processing functions required. They can be called either by one or more classes in the Presentation Layer, or even from classes that are in the Business layer itself.
- Data Access Layer: contains services/classes that enable the management of DB data. They can be called only by the Business Layer services/classes.

 ISBN: 978-1-61208-553-1

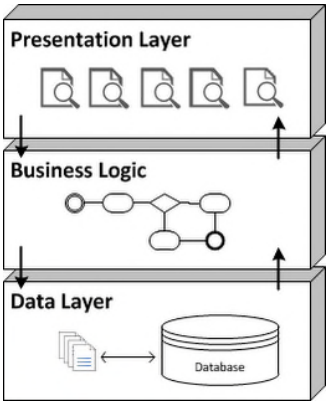

Figure 1. Three-Layers Architectural Scheme

In fact, this scheme highlights the distribution and relationship of client and server components on the specific physical nodes of an information system network of data processing.

But this kind of elements is not relevant and useful for the identification of the software objects to be measured from the user's (business) point of view. From the final user point of view, a typical elementary process (also known as Base Functional Component in the ISO standards for Functional Measurement [2]) normally begins with the business user that activates a functionality (i.e., a trigger to collect information for searching or writing data) handled by the Presentation Layer. This action activates specific features of the business logic (Business Layer) and, according to the business rules, it executes the steps needed to fulfil user requests, generally through accessing or writing permanent archives (Data Access Layer). The scenario ends crossing the layers again (in the opposite direction) to show the results to the requesting user (or to other destination users) through the Presentation Layer features. This set of steps, which are considered by the user meaningful and self-contained as a whole, crosses the layers previously identified several times.

This means that a partitioning of software application in such a way doesn't allow the proper identification of the right software items to be measured in the user functional perspective.

III. MODEL FOR FUNCTIONAL MEASUREMENT OF COMPONENT BASED ARCHITECTURE

A more usable model for a functional measurement is shown in Figure 2.

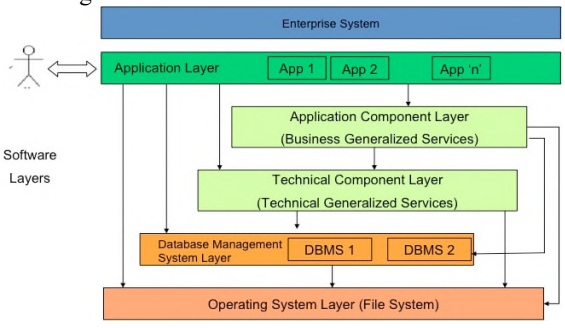

Figure 2. Multi-Layer Model for a Functional Measurement point of view

In the diagram, we see that an enterprise system can be considered as an interface for the activation of a set of applications that are available for the users in a multi-channel way and that rest, in turn, on various underlying software layers, each of them providing "services" to the above layer in a direct or indirect manner.

To clearly identify the entire domain of what to measure, from the user's point of view, we introduce the concept of Measurable Software Application (MSA).

A MSA is defined as "an aggregate of specific functional features conceived from a user point of view, homogeneous in terms of level of logical abstraction."

The term "Measurable" is necessary since very often, as we have already seen, the term Application alone is often already associated (in organization's catalogues) to groups of functionalities aggregated on a technological base instead of a logical one.

For example, in a web environment we may distinguish between a client application (browser), a data base server application, a Content Management System, a generalized "log on" feature and a library of components/web services. From the user's point of view, all these pieces are technically cooperating to support a Business Application at a logical level that is unitary in the user's view. This aggregation is called MSA.

A layer is linked, therefore, to a certain level of abstraction in the representation of data and related functions, this, in turn, determine a different concept of user associated with that particular layer.

Any MSA, by definition, should lie down on a specific and unique layer. An MSA may belong to one and only one layer so the measurement is consistent in terms of level of abstraction and it does not depend on the internal modular organization but only on the external functional user requirements.

In Figure 2, the arrows show the "call direction" of the components on the underlying layers. Between the typical Application layer and the Operating System one or two intermediate layers have been inserted: the layer of generalized business components and the one that includes generalized technical components.

The former are business functions recognizable by the user at the application level, but not sufficiently independent to be considered part of the upper level (otherwise such these business functions would be recognizable as further MSAs).

In fact they represent a kind of "recognizable pieces" of software that need to be "composed" and "aggregated" together in order to fulfil a unique user need (i.e., a component for the verification of a Security Social Number, to be inserted in several elementary processes of the Application component layer).

The latter are technical generalized functions that enables features for the management of applications (such as print programs, or a piece of software for the design of generic form, as well as a physical security service, a network service, access identification and management or client-server services too).

So, any MSA might incorporate and execute components that are distributed across multiple layers, each one containing generalized software (technical or business) components

 ISBN: 978-1-61208-553-1

designed to give specific support to the implementation of reusable and specific functional or non-functional requirements.

Therefore, in this model, a middleware layer contains a set of functions defined by the "software architect", working to aid specific requirements of factorization and independence from hardware, operating systems and DBMS environments.

As the middleware functionalities are generalized, they can then be used by different MSA (Figure 3), even not initially considered in defining the technical architecture layers.

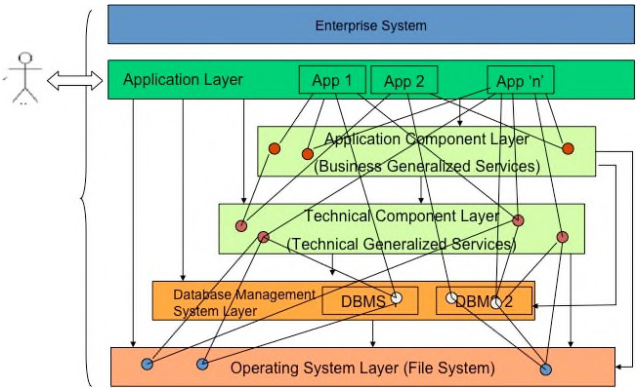

Figure 3. Link between architectural components

The IFPUG Function Point Analysis [7] requires the measurements of the software functionalities as recognized by the final user, while the functionality provided by the middleware are usually not perceived by the user him/herself, although he/she takes advantages of their presence in the systems. For example, a logon transaction for authentication of authorized users of a system can be considered as an EI (External Input) from the user's point of view since, from the software designer point of view, there may be many other elementary functions and/or intermediate software transactions, performed by the middleware and necessary for the completion of authentication service.

The functional requirements can be represented in the system specifications at different, and often not entirely consistent, conceptual and decomposition levels.

In this case a mapping activity can be necessary to correctly assign the functional user requirements (FUR) among the different software layers, to identify the software components to be measured independently each from the others.

Information exchange between different layers may be modeled and measured but different layer's measures can only be cumulated for managerial or contractual reasons. Given a certain MSA, its unique size is calculated on a single specific layer defined by its users.

In other words, the baseline FP measure of an MSA (useful, for example, to define specific service level agreements) must not be obtained as the sum of single measures performed on different layers.

It is possible to measure components at a lower level or macro functions at a higher level if we do not add their values in the asset evaluation for that specific MSA.

For example, a measurement on a Technical Generalized Services Layer can be performed to reward the development of middleware components that the supplier needs to design and build to fulfil non-functional user requirements or technological constrains that cannot be satisfied by standardized commercial-off-the-shelf solutions.

The identification of generalized software components that belong to lower level layers with respect to the business one is also crucial to quantify the rate of reuse that should be computed in each project measure for the right calculation of compensation, as defined in the contract.

The previous concepts are incorporated into the Simple Function Point specification [8], which is a more recent Functional Size Measurement Method with respect to the IFPUG one.

IV. HOW TO CONSIDER EMBEDDED SERVICES PROVIDED BY OTHER MSA

Sometimes it can be necessary to perform a new software development or a software enhancement of an MSA including elementary processes that use services of other MSA that have to be developed, modified, cancelled or remain unchanged for this purpose.

Figure 4. shows the various functional elements introduced so far to help in the understanding of what and where counting in such cases. A software development or enhancement of an MSA (MSA01 in the picture), which involves the add, change or delete of common services of another MSA (MSA02 in the picture), will count:

- within the MSA01 domain, the user functionality required;
- within the MSA02 domain, the common services that MSA02 "published" to make them available for other MSA that are affected by add, change or delete operations in order to implement the functional user requirements of MSA02.

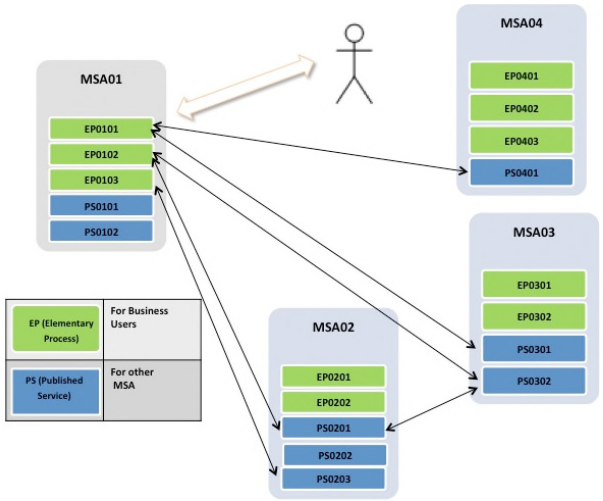

Figure 4. Relationships between MSA's components

The main cases that may occur are listed below. Suppose that an elementary process EP0103 (that is part of MSA01) uses the "published service" PS0203 (that is part of MSA02).

The most relevant scenarios are:

1) ADD of Elementary Process EP0103 (within MSA01) involves ADD of "Published Service" PS0203 (within MSA02)

2) ADD of Elementary Process EP0103 (within MSA01) involves CHG of "Published Service" PS0203 (within MSA02)

3) ADD of Elementary Process EP0103 (within MSA01) involves doing nothing for "Published Service" PS0203 (within MSA02)

4) CHG of Elementary Process EP0103 (within MSA01) involves ADD of "Published Service" PS0203 (within MSA02)

5) CHG of Elementary Process EP0103 (within MSA01) involves CHG of "Published Service" PS0203 (within MSA02)

6) CHG of Elementary Process EP0103 (within MSA01) involves DEL of "Published Service" PS0203 (within MSA02)

7) CHG of Elementary Process EP0103 (within MSA01) involves doing nothing for "Published Service" PS0203 (within MSA02)

8) DEL of Elementary Process EP0103 (within MSA01) involves CHG of "Published Service" PS0203 (within MSA02)

9) DEL of Elementary Process EP0103 (within MSA01) involves DEL of "Published Service" PS0203 (within MSA02)

10) DEL of Elementary Process EP0103 (within MSA01) involves doing nothing for "Published Service" PS0203 (within MSA02)

11) Doing nothing for Elementary Process EP0103 (within MSA01) involves CHG of "Published Service" PS0203 (within MSA02)

As an example, let's concentrate on the first case:

1) ADD of Elementary Process EP0103 (within MSA01) involves ADD of "Published Service" PS0203 (within MSA02)

In this case the elementary process EP0103 in MSA01 is created simultaneously with the creation of a common service (PS0203) in MSA02. From the perspective of MSA01, the measurement of the elementary process EP0103 is reduced because of the advantage (in terms of effort savings) earned using components "published" by the service PS0203.

This lowering is not applicable in case of baseline measurement that is a measurement for asset evaluation purposes.

From the perspective of MSA02, the service PS0203 was not pre-existing, so it is completely counted and charged as ADD, and classified as a middleware elementary process.

As in this case it is possible that some new functionalities can be built starting from pre-existing software component, allowing to obtain a significant reuse. In such cases it's possible to adopt a "lowering by reuse", applying a reducing factor (50% for example) to the data and/or transactional functions that are reusing such components.

The reuse of logical entities within a new software development means the use of logical archives already implemented in other MSAs. In this case it's no necessary to design and maintain a new data structure in the database, cause it can be reused an existing one.

This is a constraint that must be followed mainly to assure project and data integrity, and that has a consequent impact on the right calculation of FP functional size.

For transactional functions the reuse may be related to the integration of common services and application components defined and made available by corporate software frameworks.

A common service can be a set of architectural classes, support and/or shared services that have to be specially used to standardize the software behaviour and to obtain the same technical solution for some common application issues.

Any "lowering by reuse" should not be carried out in case of baseline measurement (that is a measurement for asset evaluation purposes) but only to determine the right size of new software development and enhancements.

Using this approach it is possible to allocate an adequate amount of size to the appropriate development teams that are responsible to maintain different MSA. The total size recognized for the development or enhancement task is so still consistent with an external user point of view but, at the same time, it is useful for management of different teams or even suppliers and contracts.

In an analogous way, it is possible to deal with the other 10 cases listed before.

To be clear with IFPUG experts, we are not proposing a change in IFPUG counting rules but a smart usage of the delivered size measure which becomes "worked or workable" size measure useful for managerial goals. If we look at the released functions, the standard IFPUG measure (without reuse impact) gives a size value to the end user consumable solution. The new measure (Corrected or Contractual Functional Size) is closer to the "worked size" which may be very different to the "usable size". Existing productivity rates (like ISBSG data) may be used on the Corrected Functional Size more consistently because are non "polluted" by the reuse factor.

V. COMPARISON WITH RELATED WORKS

Measuring Function Points of a SOA software application has been frequently approached using a traditional way [9][10][11].

The most used approach consists in setting the boundaries of the "objects" to be measured between the "calling" software and the "called" services, separating them and measuring the interactions among them as if they were "peer to peer" applications. This may be useful to assign to a single (used) service a sizing weight on its own but it is confusing when we consider the "calling" software that, in addition to its usual "end user interactions", has to add functionalities to deal with the usage of lower "incorporated" components in order to release "end user" transactions. This may easily lead to an over-measurement of functional size which is in contrast with the idea that software developed in a SOA environment should be "smaller" than software developed in a traditional way because of reusable component. If we have a library of reusable components we need to develop less functionalities "by scratch" and we would like to express this "saving" quantifying the software size that is reused and the size which remains to be developed completely. In the model presented in this paper, we approach the measuring of a SOA based application as a situation of component reuse: the delivered external size of the application (the released functionalities) is calculated as usual and it is not dependent on its internal architecture. In addition to this size we may calculate e second size - called "worked" or "contractual" size - which can be

 ISBN: 978-1-61208-553-1

more useful for estimation goals. Single services may still be measured separately in the traditional way if we like to manage them by metrics.

VI. NEED FOR FURTHER RESEARCH

In order to confirm the validity of the proposed approach for software governance goals an empirical research is needed to provide evidence that effort and costs may be correlated to the functional size so configured.

VII. CONCLUSIONS

The approach presented here, to measure software applications organized by the use of SOA architectures, is consistent with the ISO/IEC 14143 requirements but, at the same time, it might be useful to manage distributed efforts in software development and enhancement processes and contract management.

REFERENCES

[1] The Open Group, SOA Source Book (First Edition), Van Haren Publishing, Apr 2009

[2] A. Schmietendorf and R. Dumke, “Guidelines for the Service-Development within Service-oriented Architectures”, SMEF2007, 2007

[3] P.C. Clemens, “Software Architecture Documentation in Practice”, SEI Symposium, Pittsburgh, 2000

[4] OASIS SOA Reference Model Technical Committee, “Reference Model for Service Oriented Architecture 1.0 OASIS Standard”, 12 October 2006, Organisation for the Advancement of Structured Information Standards, https://www.oasis-open.org/standards#soa-rmv1.0, [retrieved: Mar, 2017].

[5] T. Erl, “Service-Oriented Architecture – Concepts, Technology, and Design”, Prentice Hall/PearsonPTR, 2006.

[6] ISO/IEC 14143-1:2007, Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definition of Concepts, February 2007

[7] International Function Point Users Group, “Function Point Counting Practices Manual - Release 4.3.1”, January 2010.

[8] SiFPA, "Simple Function Point Functional Size Measurement Method - Reference Manual SiFP-01.00-RM-EN-01.01", http://www.sifpa.org/en/index.htm, [retrieved: Mar, 2017].

[9] L. Santillo, `Seizing and sizing SOA applications with COSMIC function points', Proc. Fourth Software Measurement European Forum, (SMEF 2007), May 2007, Roma, Italy.

[10] J. Lindskoog,: Applying function points within a SOA environment, IFPUG Proceedings ISMA4, http://www.ifpug.org/Conference%20Proceedings/ISMA4-2009/ISMA2009-20-Lindskoog-APPLYING-FUNCTION-POINTS-WITHIN-A-SOA-ENVIRONMENT.pdf, [retrieved: Mar, 2017]

[11] Y. M. P. Gomes, "Functional Size, Effort and Cost of the SOA Projects with Function Points", Service Technology Magazine, Issue LXVIII• November 2012

 ISBN: 978-1-61208-553-1