SbSAD: An Integrated Service-based Software Design Framework

Mohamed Dbouk[1], Hamid Mcheick[2], Ihab Sbeity[1]

**[1]Faculty of Sciences-I, Lebanese University
Beirut, Lebanon**

**[2]Dep. of Computer Science, University of Chicoutimi
Quebec, Canada**

Abstract

Phased software engineering process continues to be the most popular paradigm leading to devise and drawing-up all system architectural designs. In this paper we trying to explore and examine the most significant software engineering activity: Software architectural design.

In this paper we discuss and evaluate an integrated service-based (the common and modern architectural styles upon which many systems are currently based) software architectural design framework called SbSAD. SbSAD is, mainly, built on top a proprietary micro-phased design process. In this paper, we first reconsider and refine such process in order to become more flexible. We, then, trying to evaluate this process by providing one devoted CASE-like prototype built using java technologies.

Our approach consists of building overall software architectures while being based on the concept of business front-end services. The experiments show that: applying such strategy may cause some confliction with the so known SOA and may disorient both readers and designers. However, at the end, we testify that our service-based process should not have any direct connection with the SOA style. Working on some re-drawing and mapping rules leading to transcript SbSAD into SOA could characterize our future works.

Keywords: *Software architecture, Front-end services, SDLC, SOA, CASE tools, Data exchange, Dataflow.*

1. Introduction

Software architectural design and modeling persists and remains a crucial discipline, new and additional researches are reported each day. Researchers are incited and encouraged by the newest computer-related technologies and engineering policies.

The most recent associated researches are concerned by topics like: ontological-based software architectural design, meta-modeling design and system distribution policies and strategies.

Rather than the so known client/server software architectural style, SOA (Services Oriented Architecture) plays a factual and useful software engineering implantation strategy.

However, by coming back to the most popular phased software development life cycle (SDLC), we observe that many efforts could also be deployed not only at software detailed design and modeling levels, but also at top software architectural design level.

The challenge is, in fact: could new software engineering approaches benefit from accurate and fundamental concepts, techniques and technologies like software service-based global architectural design [11] and or like service-oriented architectural style.

In other words, the software engineering experiments show that, in addition to spread out SDLC classical activities, an analytical concentration on software front-end services could perfectly help in producing an accurate software overall architecture.

For this purpose, we started by concretizing such software architectural design philosophy with our service-based architectural design approach [11].

In this paper, we start by reviewing and reformulating the advocated above approach. We focus, then, on providing an empirical framework prototyping such approach. The framework consists of a CASE tool built on top of Java facilities and putting to gather: one devoted graphical user interface and one meta-modeled repository. By incorporating such meta-data, the tool forms an open and integrated CASE framework; towards multi-platforms code generator.

On the other hand, the experiments show that, when we approach the popular software element "Service", an amalgam so arises between: the concept that we have introduced "Service-based Architectural Design SbSAD" and the so known SOA (Service-Oriented Architecture) style.

Briefly, the SbSAD that we propose and we demonstrate/prototype seems as an integrated framework intended to help in software architectural design activities.

Section 2 of this paper draws-up the associated concepts and basis. We, then, outline the related works. Principals that differentiate our approach from other approaches are also discussed in this section. Section 3 devises and reconsiders the foundations of our service-based approach. A devoted prototype (with experiments) is also outlined in this section. Section 4 tries to depict the differences between Service-based and Service-oriented terminologies.

Finally, the conclusion and the future directions are outlined and drawn in section 5.

2. Background and Related works

A computer system exists within an environment and has characteristics such as Boundaries and Front-End Interfaces. Building high-quality software is not an easy task; a wide range of software engineering paradigms have been proposed (e.g. object-orientation [3], design patterns [13] and software architectures [5]) either to make the engineering process easier or more flexible.

System development methodologies evolved [14], we depict the following progression: SDLC - Systems Development Life Cycle [17], Structured Analysis and Design (SA&D) using data-flow diagrams, Data-Oriented Methodology using Entity-Relationship diagrams and Object-Oriented Methodology using UML facilities. Where the current trend is to use Object-Oriented Systems Analysis and Design, but many organizations are still using SA&D.

Software design modeling techniques, that span stages in software lifecycle, are not standardized yet. The majority of modern software architectural strategies institute so-called decomposition methodologies. All these strategies employ some restricted vocabulary such as component (sub-system) and inter-components relationships. Goodness and robustness of the outlined software architectures are proportionally linked to the designers experience and maturity.

Even so, computer system design is concerned about the overall structure of the system [4]. How is it broken into pieces? How do these pieces fit together? The best system design is one where the interaction between the subsystems is minimal. Data Management and designing end-user interfaces are vital. These steps involve and require robust software architecture and design knowledge.

The experience of the design team, availability of pre-designed components, capabilities of design automation tools, and the maturity of the process technology, all influence the degree to which an intended computer system must be decomposed. Good and modern designs do not ignore the past [18].

On the other hand, as reported in [16], ontology can be very useful in software engineering projects where development is focused not just on one application, but on a family of projects from the same domain. Ontology must be developed in a new taxonomy framework to describe the new concepts, properties, and relationships of the new project domains.

In summary, the experience and maturity of system designers play a crucial and major role in system analysis and design process. An analysis by analogy could support most system decomposition activities.

However, the decomposition process is typically conducted by designers based on their intuition and past experiences.

The decomposition approach proposed by [20] tries to apply the clustering technique to support decomposition based on requirements and attributes. The approach supports the architectural design process by grouping closely related requirements to form a subsystem or module. Obtained decomposition and architectural styles or patterns are useful for developing a "conceptual architecture" as a representation of high-level design with critical components and connectors.

On the other hand, [12][9] propose an approach using Ontology. The idea is to close the gap between requirements and components; they use semantic models a common language ODL (Ontology Design Language) for describing product requirements and component capabilities and constraints.

Y. Cai and S. Huynh, From Drexel University in Philadelphia-USA, develop a *Logic-Based Software Project Decomposition* design representation called an Augmented Constraint Network (ACN) [6][7][8], they use the prototype tool "Simon" [1] to automatically decompose a big ACN into a number of smaller sub-ACNs.

Closely, P. Koopman proposed an elegant taxonomy of decomposition strategies [18]. The approach uses three attribute categories: *Structures* typically answer the question of "what", *Behavior* typically answers the questions of "how" and "when", and *Goals* as emergent design properties that satisfy the intended needs.

To summarize, a common theme in this discussion and our approach is a fundamental coalition between requirements, attributes and clustering techniques.

Many people have explored auto-clustering approaches to decompose an enormous dependent model into modules, such as Mancoridis's Bunch tool [21], which is based on heuristic fitness function.

There are more works done in the same context as our approach, such as the feature-oriented research led by [2], [10] and [15].

The theoretical decomposition strategies, shown before, draw some interesting and useful decomposition methodologies when they refer to requirements. These strategies, also suffer from overcrowding, from the beginning, of the deployed information (attributes).

Based on requirements and attributes, [20] applies the clustering technique to one huge and complex requirement/attribute matrix. The complexity is due to the early exploitation of information details! Otherwise, in spite of seniority and high abstraction level, the decomposition strategies drawn in [18] stay so elegant and plausible, but unfortunately, there is no pursuit.

In addition to the above talk, the recent approach ([12] and [9]) using ontology represents an innovative direction using semantic models to describe both requirements and component. By contrast, the approach predicts a component specification, things that are not plausible in our case, because depicting components is a final goal for any design process.

Lane [19] is similar to the logic-based approach [6][7][8] that models the structure of software systems as design spaces, they focus on functional choices. The logic-based approach works at abstract design level and applies formal modeling same as our approach. By contrast it applies an automatic analysis.

Finally, the architectural design approach that we propose and reconsider defers from the above approaches by many things. We introduce the concept of software design contextual dimensions (business features): profiles, services, data and rules that should characterize any front-end services. We, especially, focus on software services because they, legally, represent the only visual and interactive software entry-points. We, also, consider non-atomic data; the experiments demonstrate that there are no real needs to know details about data from the beginning.

Moreover, we continue to materialize the software engineering design scope of our approach by providing an emergent, integrated and open CASE-like framework SbSAD.

3. Service-based Design approach

The approach, also called SbSAD (**S**ervice-**b**ased **S**oftware **A**rchitectural **D**esign), which we are going to reconsider, occupies the first and crucial design stage in any traditional SDLC (fig. 1).

This approach is a micro-phased process; it is constituted around five successive analytical and modeling micro-phases. The process takes, as input, a well written (requirements) document, and produces an "Overall Software Architecture" [11].

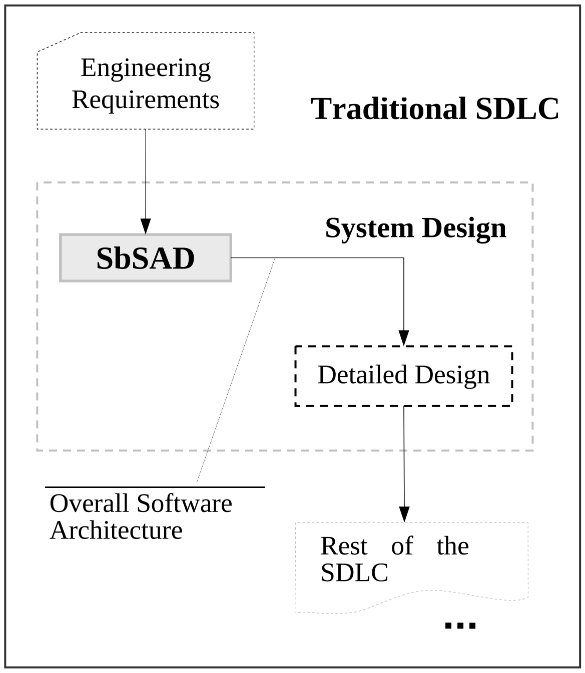

Fig. 1 SbSAD within a traditional SDLC

The approach mainly focuses on computer business front-end services, the unique visible entry-points from any software system.

3.1. Formal and algebraic definitions

This approach deals with three fundamental design features (called business/contextual dimensions) that characterize computer business services; business profiles, business data-items and business rules.

Software engineering vocabulary - the process uses the following business features:

- System (the target); computer or information system under design.
- BService (BS); front-end business related software service, materializing an entry-point.
- BProfile (BP); business domain materializing coherent set of computing activities.
- BData (BD); data sets required by the computing activities.
- BRule (BR): implicit and/or explicit business and constitutional rule, pre or post depicted against the engineering activities.

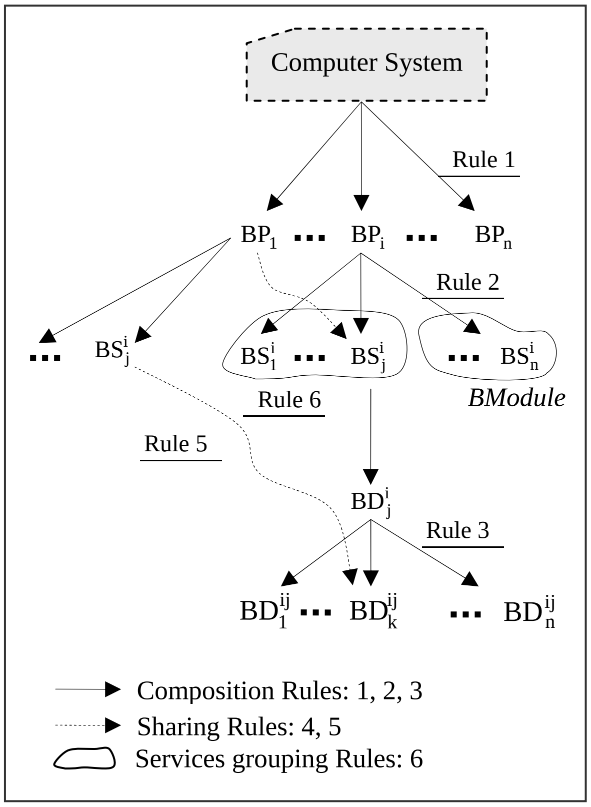

Fig. 2 Computer system structural form in SbSAD

Basically, the SbSAD process leans, indeed, on some basic constitutional architectural rules. These rules are qualified as algebraic:

Structural rules:

Rule 1- $$\text{System} \xrightarrow{\text{contains}} BP_1, \ldots, BP_i, \ldots, BP_n \quad (1)$$

Rule 2- $$BP_i \xrightarrow{\text{contains}} BS^i_1, \ldots, BS^i_j, \ldots, BS^i_n \quad (2)$$

Rule 3- $$BS^i_j \xrightarrow{\text{contains}} BD^{ij}_1, \ldots, BD^{ij}_k, \ldots, BD^{ij}_n \quad (3)$$

Engineering rule:

Rule 4- BProfiles may share BServices, formally:

$$\exists\, BS^i, BS^j \text{ where } BS^i \cong BS^j \Rightarrow BP_i.BS \cap BP_j.BS \neq \phi \quad (4)$$

Benefits: Engineering reusability of services

Behavioral rule:

Rule 5- BProfiles may share BData. Sharing issue may be direct or indirect, by similarity (same data) or by aggregation (ETL like method), formally:

$$\exists\, BD^i, BD^j \text{ where } BD^i \cong BD^j \Rightarrow BP_i.BS^i.BD \cap BP_j.BS^j.BD \neq \phi \quad (5)$$

Benefits: Business workflow depiction.

Extra-Structural rule:

Rule 6. BServices for one BP may be regrouped by sub-BProfiles in order to form the so called Business Modules BM (BModules).

Benefits: incremental system engineering building.

3.2. The SbSAD functional process

By revising the process, we observe, that there is no real need, to suddenly depict SbSAD's huge amount of conceptual features from the beginning. Instead, it will be more efficient to proceed incrementally, feature by feature.

The experiment indeed, shows that the micro-phases of this process may be applied freely; activities must be reiterated until exhausting all features.

The experiment also shows that the identified service-clusters (subsystems/components) would be reexamined in terms of modules; a module regroups one named homogenous set of front-end services and shares the same data-items with other modules inside one named business profile (Services' cluster).

The system requirements "R" represent the main source of information; the following engineering design activities might stimulate the above process:

a. read/analyze R, depict/identify one or more Business Profiles BP.

b. read/analyze R, depict/identify one or more Business services BS.
c. read/analyze R, depict/identify one or more Business Data BD.
d. explore BS, attach the BS to BP.
e. explore BD, attach the BD to BS.
f. read/analyze R, expand BD, depict/identify BDexp
g. explore BP/BD, BDexp, depict/identify BP inter-relationships, associate BProfiles mutually.
h. explore BP/BS, depict/identify Business modules BM, de-attach BS from BP, attach BS to BM and BM to BP.

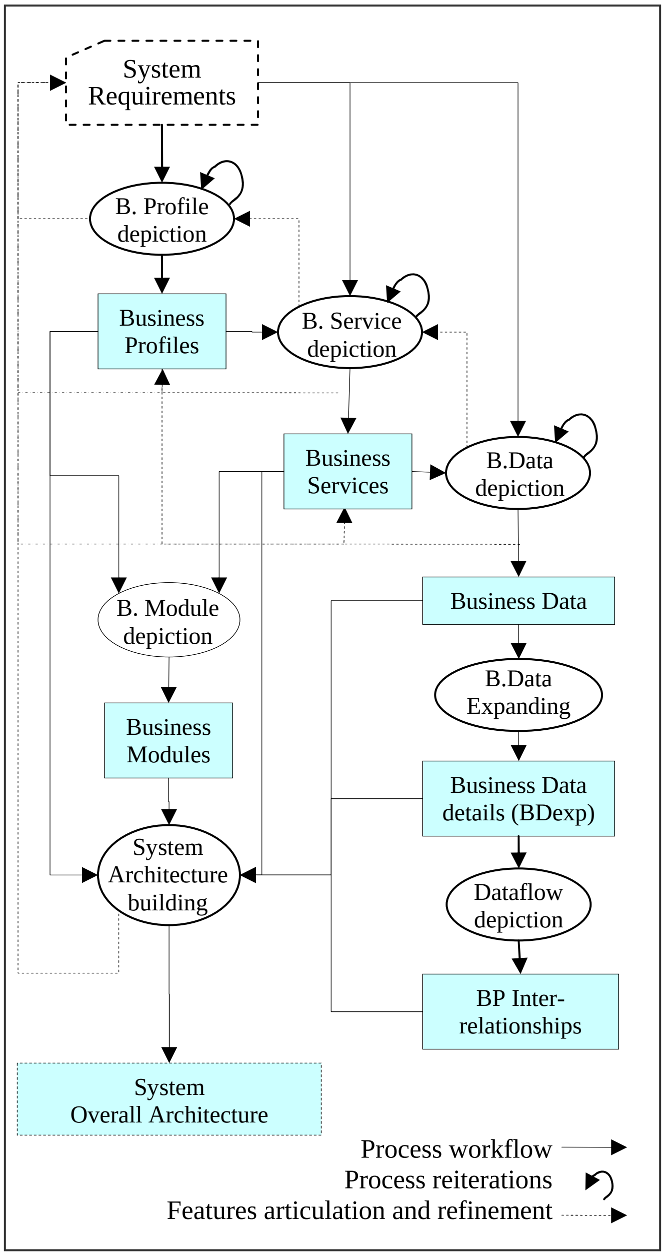

Fig. 3 SbSAD process pictogram

If we examine the above activities, we can predict that the order is insignificant, most of them may be permuted; swaps are permitted with respect to the dependability criteria, they could be performed individually.

For example, designer may perform a (non complete) sequence like this: “a, b, d, c, g, e” or this: “b, a, d, c, b, e, g”, etc. We observe that activity numbered “d” depends on “a” and “b”, etc.

However, the best way to represent such engineering design activities is by using an overall pictogram (fig. 3).

Designers can, then, build the overall architecture incrementally same as the case if we use an appropriate editing-tool (software overall architectural builder). At the end, the required SbSAD’s features should be situated and totally explored.

Designers, charged to elaborate a system design, start by reading the requirements document. They depict conceptual features one by one. Each time they identify one feature (business profile, and/or business service as well as business data), they could articulate/associate it to the adequate partner (structural rules). Designers could refer to the requirements document each time they try to perform one design activity (ovals in fig. 3).

Finally, the above pictogram materializes, transparently, all SbSAD predicted architectural rules, and produces, at the end, the intended architectural structure/design (fig. 2).

3.3. Prototyping and validation

Practically, SbSAB is in the course of prototyping. The validation of the above functional process is divided over tree stages:
- a GUI (as a CASE tool) materializing the operational process,
- building the related and required metadata,
- and validation via real use case.

The intended tool tends to integrate engineering technologies such as platforms related code generation.

SbSAD as a CASE tool: The issue is to provide a user friendly (fig. 4) graphical user interface materializing the different system design engineering functionalities.

The question now, is: How does such tool operate? A brief abstraction of the operational process is given in fig. 5.

However, the main purpose of this tool is to provide a useful interactive framework building and producing the overall system architecture.

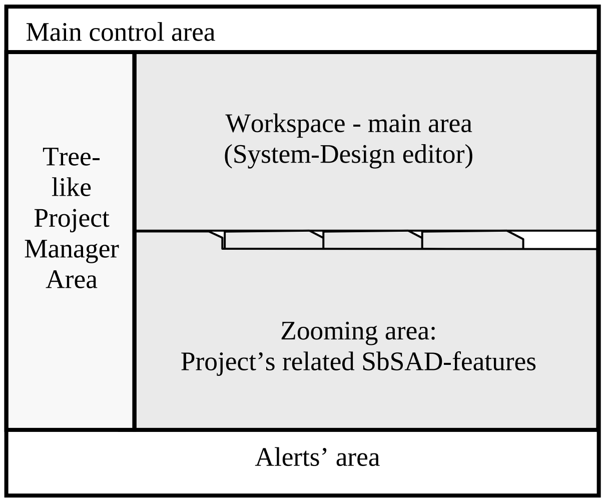
Main control area
Tree-like Project Manager Area
Workspace - main area (System-Design editor)
Zooming area: Project's related SbSAD-features
Alerts' area

Fig. 4 The SbSAD's interactive graphical interface

The tool incorporates one crucial (integrated) piece that supports the implementation software engineering phase; it refers also to one devoted back-end meta-model.

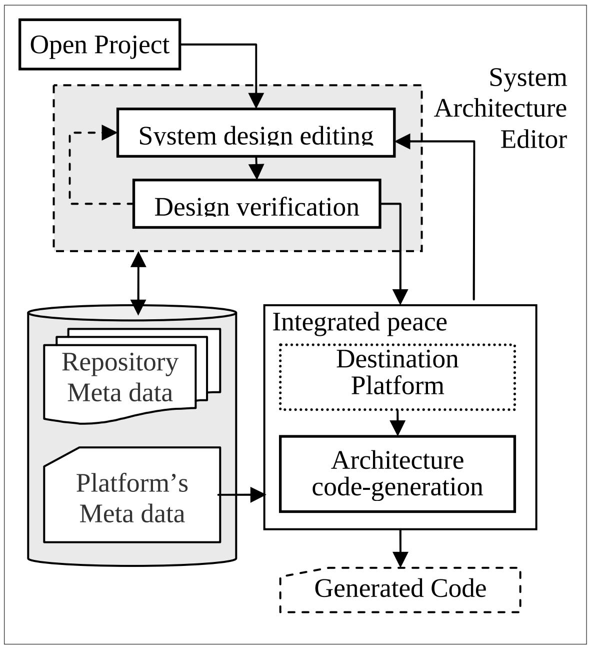

Fig. 5 SbSAD's CASE tool operational process

Back-end Meta model: The SbSAD's operational process refs to some devoted meta-data (fig. 6), a crucial piece that consists of a collection of back-end related data (enclosed class diagram).

However, the data model incorporates all SbSAD related features and concepts. In addition to information cataloging the system required data, the data-model, especially, includes the overall system structural and dataflow behavioral features.

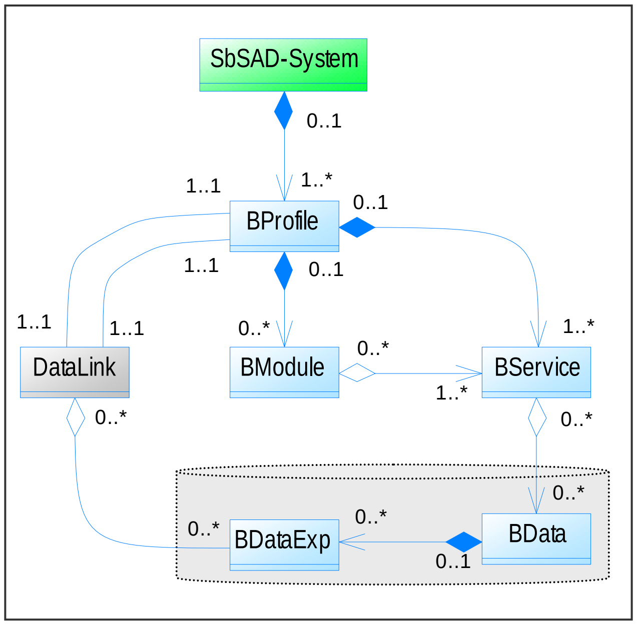

Fig. 6 Repository UML class diagram

Finally, this information forms an editable repository that could be strongly used by the tool.

Use case and validation: The first and empirical version of SbSAD is now operational; it is built on top of Java technology. Many improvements and enhancements are planed.

The enclosed figure (fig. 7) draws a typical case: Sale/Purchase-Management Information System (S/P-MIS). The given sample illustrates and shows the following (SbSAD) features:

- Business Profiles (SubSystems): Stock-Manager (StkMg), Point-Of-Sale (PoS), OLAP-Manager (OlapMg), etc.
- StkMg may incorporate: Produts-Nomination (PdNames) and Inventories (InvMg) etc. as modules.
- Intiate-PoS and Close-PoS are data-links relating StkMg to PoS considered subsystems and vice versa.

  Those links could be infected by the chosen strategy to materialize the data sharing issue; online or offline (differed) mapping. A crucial data link (ETL-like) should exist between StkMg and OlapMg.

Finally, many kinds of data items could characterize the "S/P-MIS"; products, commands, customers (if considered), suppliers, cashers, etc.

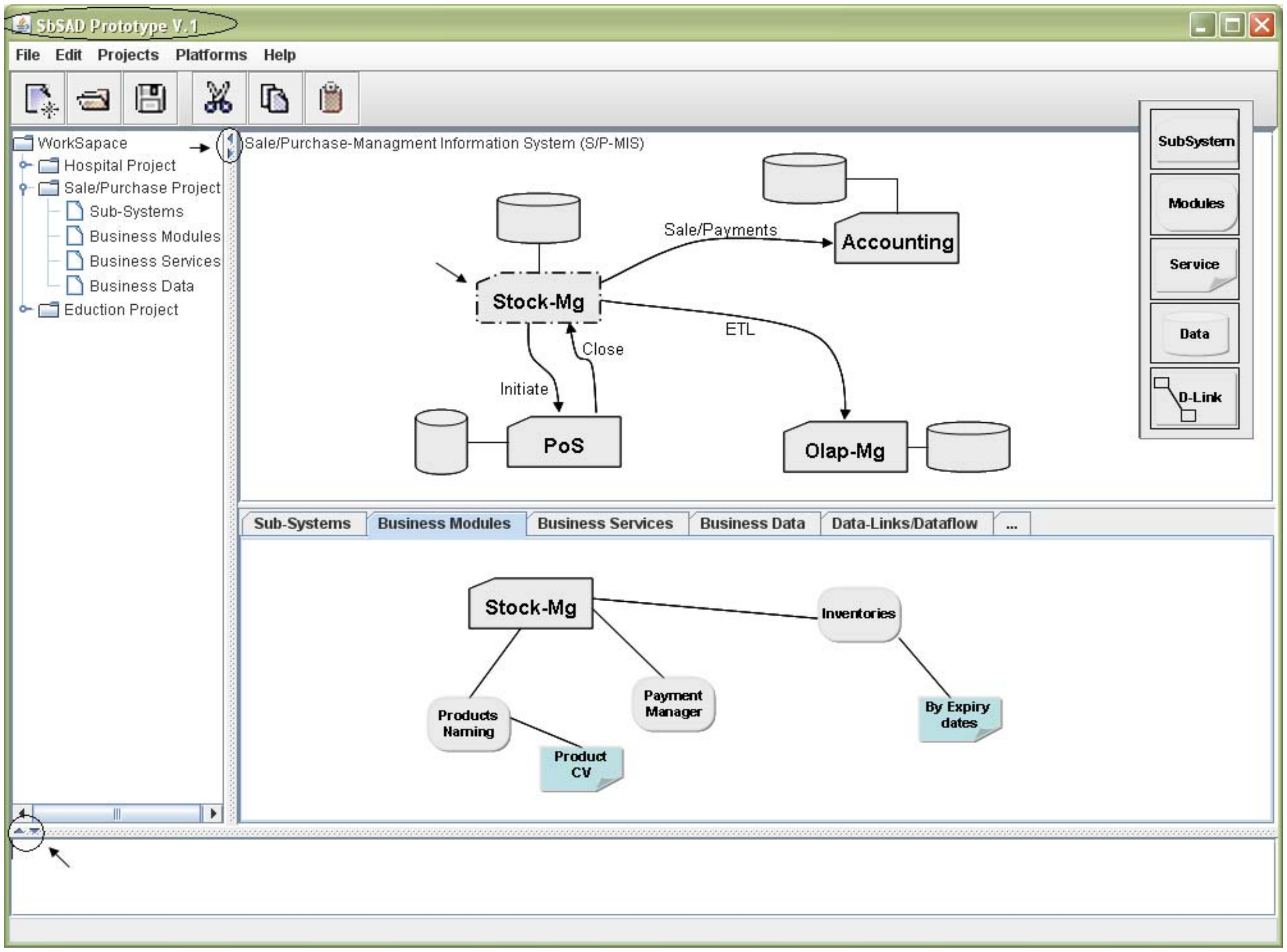

Fig. 7 The SbSAD's prototype; integral graphical user interface view.

4. SbSAD vs. SOA

As mentioned before, the SbSAD process is concerned by providing the overall system/software architecture. The process represents one of two major design phases (figures 1 & 8). At this stage, the designer doesn't have to worry about the manner, according to which the target system could be implemented and deployed. Such task, strictly, comes after.

The, so known, SOA (Service Oriented Architecture) policy and strategy is, practically, especially concerned with the implementation and system deployment issues.

So, the question that arises is: How could we qualify the connection/relationship between SbSAD and SOA?

First of all, the SbSAB approach is intended to be a design process while SOA is seen as an implementation related architectural style (fig. 8).

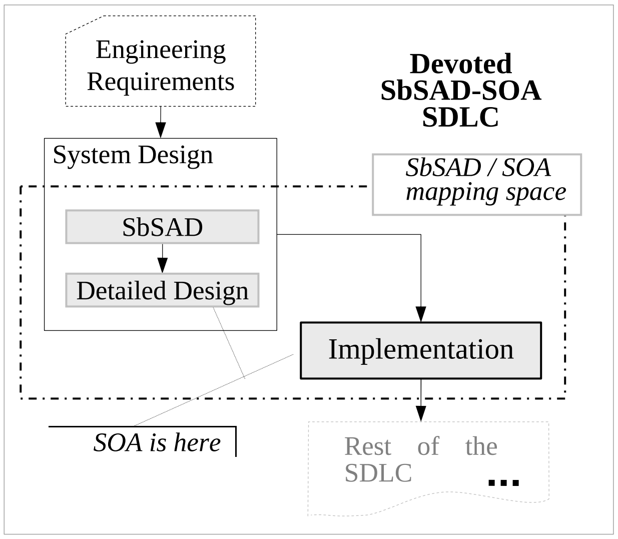

Fig. 8 SbSAD / SOA Space

Practically, a close mapping of SbSAD's outcomes into SOA methods and techniques should produce one devoted and powerful software design and implementation process. Such mapping process could include one dedicated set of rules that drawing out one regular SbSAD-SOA software engineering policy.

5. Conclusion and Future work

We started this work by establishing the main software architectural design features, context and background.

We devised, in this paper, one dedicated software engineering framework (SbSAD), it consists of an integrated CASE-like tool. The tool is prototyped using Java technology, the first version gives good results, additional features and improvements are planed.

However, we, in this paper, reconsidered and reevaluated the service-based software architectural design process. We talked, especially, about the formal definition as well as the operational modalities of the process.

In addition to the above talk, the experiments show, that the process might, easily, be extended. It could incorporate new design facilities; detailed design, multi-platform related design, etc.

To conclude, as future work, we plan to work on the mapping issues between SbSAD outcomes and SOA style.

References

[1] Baldwin, C.Y. and Clark, K.B., "Design Rules", Vol. 1: The Power of Modularity. Publisher: The MIT Press, 2000-03-15, 483 Pages, ISBN: 0262024667

[2] Batory, D., Singhal, V., Thomas, J., Dasari, S., Geraci, B. and Sirkin, M., "The genvoca model of software-system generators", IEEE Software, 11(5):89–94, Sept. 1994.

[3] Booch, G., "Object-oriented analysis and design with applications", 1994, Addison Wesley.

[4] Bruegge, B. and Dutoit, A.H., "Object Oriented Software Engineering Using UML, Patterns and Java", Second Edition. Pearson Education International, 2004

[5] Buschmann, F., Meunier R., Rohnert, H., Sommerlad, P. and Stahl, M., "A System of Patterns", 1998, Wiley.

[6] Cai, Y., "Modularity in Design: Formal Modeling and Automated Analysis". PhD thesis, Univ. of Virginia. 2006.

[7] Cai, Y. and Sullivan, K., "Modularity analysis of logical design models". In 21th IEEE/ACM Int. Conf. on Automated Software Engineering, Tokyo, JAPAN, 2006.

[8] Cai, Y. and Simon, K.S., "A tool for logical design space modeling and analysis". In 20th IEEE/ACM Inter. Conf. on Automated Software Engineering, Long Beach, California, USA, Nov 2005.

[9] Cardei, I., "An Approach for Component-based Design Automation", Whitepaper, Florida Atlantic University press 2006

[10] Czarnecki, K. and Eisenecker, U., "Generative Programming: Methods, Tools, and Applications'. Addison-Wesley Professional, 1st edition, Jun 2000.

[11] Dbouk, M., Sbeity, I. and Mcheik, H., 'Towards Service-Based Approach; Building Huge Software Architectural Designs', IJCNDS, 2011, v6 (forthcoming).

[12] Fonoage, M., Cardei, I. and Shankar, R, "Mechanisms for Requirements Driven Component Selection and Design Automation" the 3rd IEEE Systems Conference, Vancouver, Canada, 2009.

[13] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., "Design Patterns", 1995, Addison Wesley.

[14] George, J.F., Batra, D. and Valacich, J.S., "Object-Oriented Systems Analysis and Design", ISBN: 0132279002, ISBN-13: 9780132279000, Published by Prentice Hall, 2006

[15] Goguen, J.A., "Reusing and interconnecting software components". IEEE Computer, 19(2):16–28, Feb. 1986.

[16] Hesse, W., "Ontologies in the software engineering process". In R. Lenz et al., editor, EAI 2005 Proceedings of the Workshop on Enterprise Application Integration

[17] Hoffer, J.A., George, J.F. and Valacich, J.S., "Modern Systems Analysis and Design (5th Edition)", ISBN-10: 0-13-224076-9, ISBN-13: 978-0-13-224076-5, Published by Prentice Hall, 2008

[18] Koopman, P., "A taxonomy of decomposition strategies based on structures, behaviors, and goals", 1995 Conference on Design Theory and Methodology, Boston, September 1995.

[19] Lane, T.G., "Studying software architecture through design spaces and rules", Technical Report CMU/SEI-90-TR-18, CMU, 1990.

[20] Lung, C. and Zaman, X. M., "Software Architecture Decomposition Using Attributes" Carleton University, Ottawa, Ontario press, Canada 2006.

[21] Mancoridis, S., Mitchell, B., Rorres, C., Chen, Y. and Gansner, E., "Using automatic clustering to produce high-level system organizations of source code". In Proceedings of the 6th Inter. Workshop on Program Comprehension (IWPC'98), pp. 45–52, June 1998.

Mohamed DBOUK, received a Bachelor's Honor" in Applied Mathematics; Computer Science, Lebanese University, Faculty of Sciences, and a PhD from Paris-Sud 11 University (Orsay-France), 1997. He is a full time Associate Professor, at the Lebanese university - Faculty of Sciences-I, Dep. of Computer Science. His was (2005-2007) the director of this Faculty, and he is the Founder and Coordinator of the research master "M2R-SI: Information System". His research interests include Software engineering, Information systems, GIS, Cooperative and Multi-Agent Systems, Groupware. He participates in many international projects.

Hamid Mcheick is currently an associate professor in Computer science department at the University of Quebec At Chicoutimi (UQAC), Canada. He holds a master degree and PhD. in software engineering from Montreal University, Canada.

Ihab Sbeity occupies a full time position in Computer Science Department at the Lebanese University. He holds a PhD.in performance evaluation and system design from "Institut National Polytechnique de Grenoble", France in 2006