International Journal of Information Systems and Project Management

Volume 1 | Number 1 Article 3

2013

A multi-layered software architecture model for building software solutions in an urbanized information system

Sana Guetat
Le Mans University

Salem Dakhli
Paris-Dauphine University

International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

A multi-layered software architecture model for building software solutions in an urbanized information system

Sana Bent Aboulkacem Guetat
Le Mans University, ARGUMANS Lab.
Avenue Olivier Messiaen, 72000, Le Mans
France
www.shortbio.net/sana.guetat@univ-lemans.fr

Salem Ben Dhaou Dakhli
Paris-Dauphine University, CERIA Lab.
Place du Maréchal de Lattre de Tassigny, 75575 Paris
France
www.shortbio.net/sdakhli@computer.org

Abstract:
The concept of Information Systems urbanization has been proposed since the late 1990's in order to help organizations building agile information systems. Nevertheless, despite the advantages of this concept, it remains too descriptive and presents many weaknesses. In particular, there is a lack of useful architecture models dedicated to defining software solutions compliant with information systems urbanization principles and rules. Moreover, well-known software architecture models do not provide sufficient resources to address the requirements and constraints of urbanized information systems. In this paper, we draw on the "information city" framework to propose a model of software architecture - called the 5+1 Software Architecture Model - which is compliant with information systems urbanization principles and helps organizations building urbanized software solutions. This framework improves the well-established software architecture models and allows the integration of new architectural paradigms. Furthermore, the proposed model contributes to the implementation of information systems urbanization in several ways. On the one hand, this model devotes a specific layer to applications integration and software reuse. On the other hand, it contributes to the information system agility and scalability due to its conformity to the separation of concerns principle.

Keywords:
information system urbanization; software architecture; software layer; software service; architecture rule; urbanization principle.

DOI: 10.12821/ijispm010102

Manuscript received: 18 January 2013
Manuscript accepted: 4 February 2013

1. Introduction

Almost all modern organizations are faced with more pressures from the ever-changing external economic, technological, social and political environments. Therefore, they have to continuously adapt their priorities, processes products, services and relationships with their partners, customers and suppliers, in order to be compliant with new business rules and market constraints. Moreover, such organizations make today a heavy use of information and communication technologies while implementing their organizational processes. As highlighted by many authors, complexity is among the essential characteristics of organizational information systems [2], [18], [19]. Moreover, Brooks [1], [2] states that information systems have four main properties: complexity, conformity, changeability, and invisibility. In particular, information system complexity is both structural and systemic. The structural complexity of an information system is associated with its structure attributes such as the number and the size of its software applications. The systemic complexity of an information system is related to the interactions between its parts (inter-applications and intra-applications interactions) and the informational flows and services exchanged with external information systems. Information system complexity results in many problems. On the one hand, the difficulty of communication between information system stakeholders leads to poor quality, costs and delays overruns. On the other hand, the difficulty of understanding all the states of software applications, leads to maintenance and evolution problems. Finally, the difficulty of getting a global view of an information system may jeopardize its conceptual integrity. Changeability results in information system evolution. It reflects the need for an organization to continuously adapt its information system in order to take into account its business environment pressures. As stressed by Lehman and Belady [20], information systems applications that are really used change continuously because they are exposed to many forces that require them to change. Information systems evolution is governed by Lehman's law of continuing change which establishes that "*A large program that is used undergoes continuing change or becomes progressively less useful. The change process continues until it is judged more cost-effective to replace the system with a recreated version*" [21].

Software systems aging is another important characteristic of information systems. As stated by Parnas [3], like human aging, software aging is inevitable, but like human aging, there are things that can be done to slow down the process and sometimes even reverse its effects. This author uses the decay metaphor to describe how and why software becomes increasingly brittle over time, and identifies two types of software aging which lead to a decline in the value of a software system: the failure to keep up with changing environment, and the software damages caused by the software changes made. Lehman [21] considers that software aging may be related both to software complexity and software continuous change. According to Lehman's law of increasing complexity, "*as a large program is continuously changed, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it*". Software aging results in decreased performance and reliability due to the software structure deterioration and errors related to changes, and inability to keep up with the market due to increasing size and complexity. In addition to the information systems problems related to the essential characteristics of software, other problems inherent in information systems originate from their accidental characteristics [1]. In particular, many organizations have built their information systems in a chaotic manner materialized by the development and deployment – by each organizational unit – of its own software applications without taking into account redundancies and coherence with applications deployed by other organizational units. Such a way of developing software systems leads to information systems which are complicated, high resource consumers, expensive to maintain, and inflexible. In such a situation, the computerization of any change in organizational processes may be expensive since it mobilizes important resources necessary to identify and modify all the software applications that are affected. According to Perry and Wolf [15], software evolution is strongly dependent on software architecture. These authors describe architecture as the "*load-bearing walls*" of a software system which allows some degree of evolution. In other words, to remain compliant of architectural rules and constraints, software systems architecture allows some changes and precludes others which require a migration to a new architecture. Moreover, some allowed software changes may involve such a migration because they are too expensive to be implemented with the current software architecture. Perry and Wolf [15] consider that software change induces two types of architectural evolution: architectural drift and architectural erosion. The former occurs when software changes

are based on a software architecture that is different from the intended architecture. It is due to misunderstanding of the current architecture by the software developers involved in software systems evolution. The latter is caused by violations of software systems architecture and results in increased software systems brittleness. Architectural drift and architectural erosion have negative impacts on software systems maintainability and often lead to architecture redesign.

Information systems problems described in this section impede building agile customer-oriented organizations which need to be supported by open and agile information systems that can be integrated in a secure and efficient mode, with the systems of its customers and suppliers. The concept of information systems urbanization has been proposed since the late 1990's in order to help organizations building agile information systems. Nevertheless, despite the advantages of this concept, it remains too descriptive and presents many weaknesses. In particular, there is a lack of useful architecture models dedicated to defining software solutions compliant with information systems urbanization principles and rules. Furthermore, well-known software architecture models do not provide sufficient resources to address the requirements and constraints of urbanized information systems. In this paper, we draw on the "information city" framework to propose a model of software architecture - called the 5+1 Software Architecture Model - which helps organizations building urbanized software solutions. This framework improves the well-established software architecture models and allows the integration of new architectural paradigms. Our paper is organized as follows. In section 2, we recall the principles of information systems urbanization and present the "information city" framework which constitutes the theoretical foundation of our work. In section 3, we define software architecture and provide a critical analysis of the main software architecture models. Section 4 is dedicated to the presentation of the multi-layered 5+1 software architecture model. In section 5, we conclude this paper by listing its contributions and future research directions.

2. The information city framework

Information systems urbanization is a strategic planning approach for building agile organizational information systems. It includes a set of governance instruments which facilitate the scalability and strengthen the coherence of organizations information systems. This approach is based on four activities: mapping the existing information system, definition of the target information system, gap analysis, and description of the roadmap to reach the target information system. Information systems urbanization is a complex process. The complexity of the information systems urbanization results from the complexity of the information systems artifacts handled by this process. Therefore, to understand information systems urbanization, we use metaphors. Lakoff and Johnson [22] define a metaphor as a way of thinking and seeing that helps understanding one kind of things in terms of another.

The "city planning" or "city landscape" metaphor is among the most popular metaphors proposed in the academic literature to define the foundations of enterprise architecture and information systems urbanization [23], [24], [25]. The application of the "city landscape" metaphor to information systems urbanization has several weaknesses resulting from its descriptive orientation. First, it does not facilitate the identification of architectural principles and rules applicable to complete information systems urbanization. Second, it does not indicate how to manage and take into account the information systems complexity during the construction of urbanized information systems. The "information city" framework [4] generalizes the use of the "city planning" metaphor by stating that - within a modern organization - an information system may be considered as a city where the inhabitants are the applications belonging to this information system. In this city, called the information city, the common parts are software artifacts and information shared by all the information system applications while the private parts are composed of software artifacts and information owned by each application. An application belonging to the information city behaves as a master of its proper data and artifacts and as a slave regarding data and software artifacts which belong to other applications. That means that an application can use, update or suppress data and artifacts it owns but can only use a copy of other applications data and software artifacts.

Comparing an information system to a city extends the use of the "city landscape" beyond the analogy between software and building construction by emphasizing the problem of information system governance. On the one hand, following the example of a city, the relationships between the applications which populate the information city must be

managed. That means that a set of architecture principles and rules has to be specified in order to govern exchanges either between application belonging to an information system or between such applications and the external environment like other information systems or end-users. On the other hand, the vast number of application assets in combination with the natural expansion of the application portfolio, as well as the increasing complexity of the overall information system, drives a need for the information system governance. Therefore, the "information city" framework permit defining architecture principles and rules which help organizations prioritize, manage, and measure their information systems.

Using the "information city" framework makes organizations able to apply a structure for classifying information system applications, functions, or services in a coherent way. It defines responsibility plots from coarse to fine-grained into discrete areas, which together form the complete Information City Plan (ICP) (Fig. 1).

The ICP is a set of areas, districts, and blocks. An area is composed of districts and a district splits into blocks. The ICP areas are determined according to three urbanization principles resulting from a deep analysis of the organization's and information technology strategies. These principles are:

- Urbanization principle 1: Determine front-office *vs.* back-office responsibilities;
- Urbanization principle 2: Specialize front-office and back-office regarding the organization's processes;
- Urbanization principle 3: Identify the components common to the back-office and the front-office**.**

The first architecture principle - Determine front-office *vs.* back-office responsibilities - identifies the responsibilities of the organization's front-office and back-office. The front-office is dedicated to management of the relationships with the organization's external environment while the back-office is dedicated to the development of products and services. For instance, within an insurance company the back-office manages the insurance and services commitments whatever the distribution channels.

To apply the second architecture principle - specialize front-office and back-office regarding the organization's processes - we use a classification of organizational processes into five categories: business processes, support processes, decision-making processes, communication with the organization's external environment processes, and management of the relationships with the organization's external environment processes. The first three categories relate to organizational processes in the back-office, while the last two refer to those in the front-office. According to this classification, the second architecture principle permits identifying at least three areas in the organization's back-office and two areas in the organization's front-office. The back-office areas are:

- The "Business Intelligence area" associated with decision-making processes;
- The "Support area" associated with support processes;
- And at least one "Business area" associated with business processes: the "Policy and Claims area" is an example of a business area within an insurance company.

The front-office areas are the "Inbound and Outbound flows Management area" and the "Party Relationships area". The "Inbound and Outbound flows Management area" is associated with the communication with the organization's external environment processes. This area is dedicated to the management of the informational flows exchanged by an organization and its external environment. It describes the various technology channels used by an organization while exchanging information with its external environment. The "Party Relationship area" is associated with management of the relationships with the organization's external environment processes. This area supports the relationships linking an organization with its customers and partners whatever the communication channel.

The third architecture principle - Identify the components common to the front-office and the back office - refers to either the components that link the front-office and the back-office or the artifacts shared by the back-office and the front-office. The application of this principle results in identifying two areas: an "Integration area" and a "Shared information area". The first area allows exchanges of informational flows and services between the back-office and the front-office applications.

The second area contains information shared by all the applications of the organization's information system as well as the applications which manage shared information data. The customers and products repositories are examples of information shared by all the applications of an organization's information system.

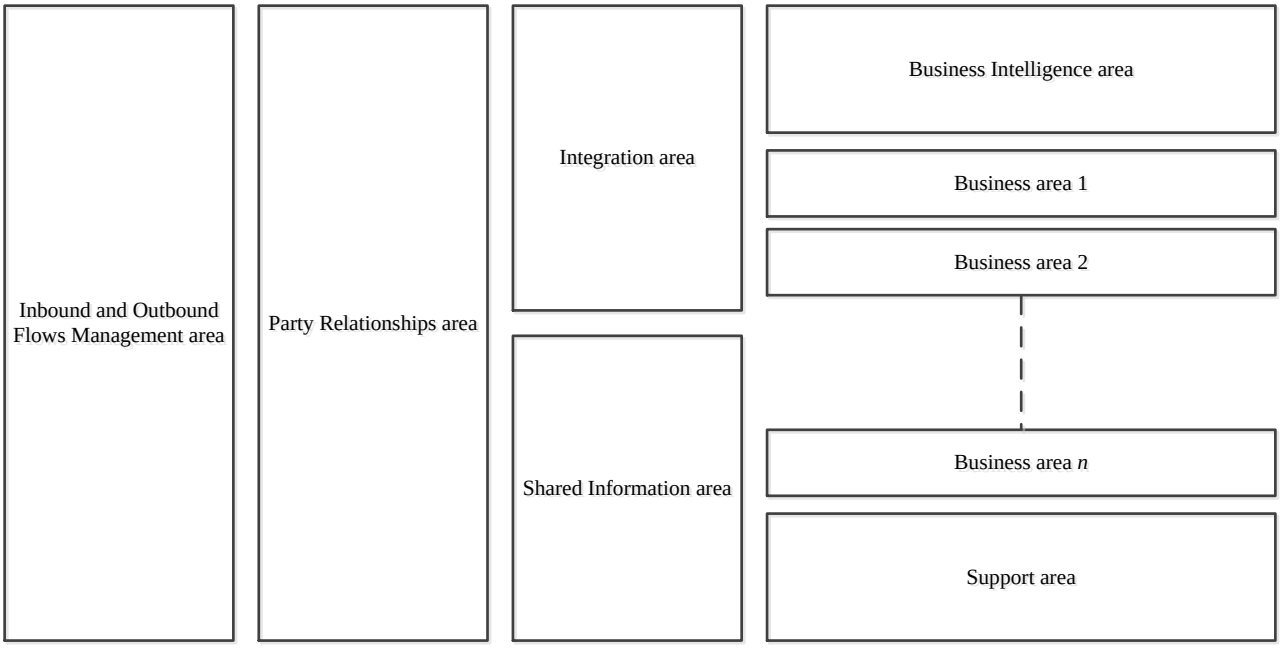

Fig. 1. The Information City Plan (ICP)

3. Software architecture: definitions and critical analysis

As stressed by many authors, software architecture has emerged as an important field of information systems for managing software applications development, evolution, and maintenance [7]-[10]. The main intent of software architecture is to provide intellectual control over a complex software system [11]. Indeed, software architecture models the structure and behavior of a system; and presents a high level view of a system, including the software elements and the relationships between them. Software architecture is a complex concept that is difficult to capture in a single definition. Many definitions of the software architecture concept are proposed in the literature. For example, Toffolon [5] and Toffolon and Dakhli [6] consider that software architecture describes a software solution which computerizes an organizational solution of an organizational problem. Garlan and Shaw [26] stress that software architecture is characterized by a set of issues which include gross organization, global control, structure, communication protocols, and assignment of functionality to design elements. Kruchten [12] and Jansen [27] draw on work by Shaw and Garlan [10] to define software architecture as the set of significant decisions about the organization of a software system. Such decisions focus on the selection of the structural elements and their interfaces by which a system is composed, the behavior as specified in collaborations among those elements, the composition of these structural and behavioral elements into larger subsystem, and architectural style that guides this organization. These authors point out that software architecture also involves usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and tradeoffs, and aesthetic concerns. Bass et al. [7] propose a definition of software architecture that acknowledges that architecture of a single software system may be described using different types of

structures. According to these authors, software architecture is based on the structure or structures of a software system, which includes software elements, the externally visible properties of those elements, and the relationships among them. The IEEE 1471-2000 standard [13] defines software architecture as the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. In addition to the software architecture components and their relationships, this definition covers architecture rules and principles like architectural styles or the use of particular conventions during the software development, maintenance, and evolution life cycles.

Due to the complexity of the software architecture concept, two problems have to be solved in order to understand it and apply it efficiently in software development, maintenance, and evolution projects: software architecture description, and software architecture modeling. The first problem means that it is difficult to describe software architecture by one structure, or by one type of abstraction. To deal with this problem, academics and practitioners suggest using views and perspectives. To this end, Kruchten [28], Hofmeister et al. [29], and Clements et al. [30] stress that multiple views are required to completely describe and document software architecture. Each view of a software architecture addresses a specific set of concerns and is created using guidelines defined in a viewpoint.

According to Jansen et al. [14], software architecture has three perspectives: blueprint, roadmap, communication vehicle, and quality predictor. The blueprint perspective outlines software systems structures and behaviors. The roadmap perspective describes the evolution paths of software systems. The communication vehicle perspective focuses on the communication instruments used by the software system stakeholders to share the architectural decisions in order to steer and influence the design of a software system. The quality predictor perspective provides an early predictor of the quality of software systems architectures. As noted above, software architecture is inevitably subject to evolution due to software aging due to architectural drift and architectural erosion [15].

To solve the second problem and support information systems evolution without compromising their invariants and integrity, many software architecture models have been proposed by academics and practitioners such as the Perry and Wolf's architecture model [15], the product line model [31], and the multi-layered architecture like the Model-View-Controller (MVC) [33], the Presentation-Abstraction-Control (PAC) [16], and the multi-tiers architecture models. In particular, the architecture model proposed by Perry and Wolf [15] consists of design elements, form, and rationale. Firstly, design elements include data, processing, and connecting elements. Secondly, forms refer to the relationships among the elements of software architecture. Finally, rationale describes the motivation for the decisions that yield a particular set of elements and form.

The software architecture product-line model has been proposed by Clements and Northrop [31] to increase software reuse through the use of architectural rules and principles. A software product-line encompasses a whole range of software artifacts that have common characteristics. It involves the development of product-line assets, such as a product-line architecture, reusable components, and product-line members. The assets that apply to the product line as a whole are developed in a process referred to as domain engineering while the product-line members are developed in a process called application engineering. According to [31], the development of such products as a software product-line makes their commonalities and variability explicit in a product-line architecture. Therefore, the development of individual products consists in binding the variation points defined in the product-line architecture to specific instances [32].

The multi-tiers architecture model is a multi-layered software architecture model which provides a logical way to separate the different responsibilities of software applications. For example, according to the three-tier software architecture model, a software system is composed of three parts called tiers:

- The presentation tier is responsible for displaying information and supporting the interactions with the end-users;
- The application tier - also called business tier - is responsible for the coordination of the software system business logic, *i.e.*, it executes commands, actions and moves data between the presentation and the data tier;
- The data tier - also called persistence tier - is responsible of data retrieving and storage.

The software system tiers should be as independent as possible from each other. The MVC model suggests that a software system is based on three components: the Model component, the View component, and the Controller component [32]-[33]. The Model component provides the core functionality of the software system. The View component role consists in presenting models from the Model component. The Controller component manages interactions between end-users and views from the View component, and checks how views and models are manipulated by end-users. A model may be associated with many views which are notified by this model whenever it is updated. This enables developers to create or modify views without altering the associated model, and guarantees that all views associated with a model are synchronous since they reflect the same model state. Although the MVC model is different from the multi-tiers architecture model, we think that the View and Controller components belong to the Presentation tier while the Model component belongs to the application and data tiers. The same can be applied to the PAC model.

Through the separation of responsibilities, the multi-layered software architecture model facilitates the management of many aspects of the information systems complexity, and improves software systems maintenance and evolution. Nevertheless, many important problems remain unsolved. For example, these models do not provide efficient ways for cooperation either within the same information system or between many information systems. Moreover, the existing multi-layered software architecture models do not consider the constraints and rules of urbanized information systems. In other words, the multi-layered models (multi-tiers, MVC, PAC, etc.) need to be enhanced in order to manage efficiently the complexity inherent in multi-channels access and navigation, or services and information flow exchanges.

4. The multi-layered 5+1 software architecture model

An urbanized application is a software system whose architecture is compliant with the information city goals such as agility and reuse. This means that an urbanized application must meet the basic architectural rules and principles induced by urbanization constraints. “Strong coherence and weak coupling”, “separation of concerns”, “standard communication protocols”, and “data hiding” are examples of such principles. Moreover, an urbanized application should take into account the four urbanization principles used to build the Information City Plan (ICP). As a result, an urbanized application is organized as a set of parts that have public resources and private resources, and interact by using standard communication protocols. Therefore, the architecture of an urbanized application is organized in layers, each layer being responsible for a specific concern. In this section, we propose a software architecture model - called the 5+1 model - to help design urbanized applications.

The multi-layered 5+1 model, which describes the architecture of software systems belonging to urbanized information systems, is composed of six architecture layers: the Interface layer, the Navigation layer, the Orchestration and Choreography layer, the Services layer, the Data Access layer, and the Technical Services layer. Each layer is associated with a data set which describes its modifiable parameters. These parameters are stored in a read-only repository - called layer repository - which enables their update without modification of the software system programs. Fig. 2 illustrates the multi-layered 5+1 software architecture model.

4.1 Presentation of the 5+1 model layers

The Technical Services layer includes technical services shared by the other layers. Security services, network services, errors management services, and middleware services are examples of technical services managed by this layer. The detailed description of this layer is outside the scope of this paper. Table 1 provides a synthetic description of the other five layers of the 5+1 software architecture model which includes information related to the role of each layer, the functions it supports, the content of its repository, and the associated architecture rules.

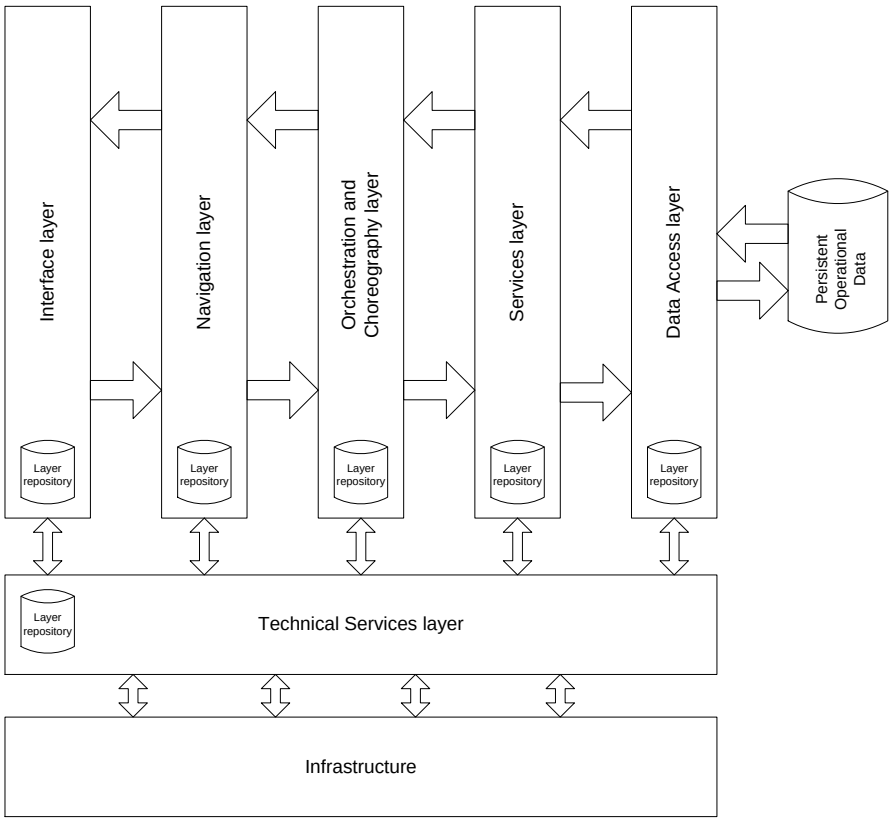

Fig. 2. The multi-layered 5+1 software architecture model

Table 1. The description of the 5+1 model layers

Layer	Role	Supported functions	Layer repository (Examples	Architecture rules (Examples
Interface	- Management the interactions with end-users for each technical communication channel - Management of the graphical aspects of the human-machine interface - Components: screens, editions, and formatting elements	-Presentation management - Screens content display - Syntax control of data - Surface control of input data - Online help	- Displaying colors - Messages labels - Screens associated with a language	- **Rule 1:** Only the modules of the Interface layer can interact with human end-users
Navigation	- Description of the progress of the screens kinematics - Management of data specific to the interaction between the software system and its human end-users - Management of a context related to informational flows exchanged with the Orchestration and Choreography layer - Components: technical communication channels kinematics (Internet, Mainframe 3270, …)	- Identification of the screens for tasks performing - Calls to the Orchestration and Choreography layer modules to carry out controls related to the organizational processes supported by the software system - Routing of displayed information to local printers (forms, display styles, …)	- List of screens associated with a task	- **Rule 1:** Only the modules of the Navigation layer can call the Orchestration layer modules

Table 1. The description of the 5+1 model layers (cont.)

Layer	Role	Supported functions	Layer repository (Examples	Architecture rules (Examples
Orchestration and Choreography	- Identification of the organizational processes activities supported by the software system - Management of the sequence of tasks supported by the software system - Description of the end-users roles - Control of the informational and services exchanges with other software systems - Management of a context related to tasks running in order to allow interruptions without data publication	- Start and complete a sequence of tasks - Expose services to other software systems - Call of services exposed by other software systems - Sequence of services invocation - Services orchestration - Services choreography	- List of tasks making up a use case - List of services used by a use case - Description of the sequence of tasks making up a use case	**- Rule 1:** The business data processed during process execution are managed in a process context. This context is managed exclusively by a module of the Orchestration and Choreography layer. No module of another layer can access it even in reading-only **- Rule 2:** Only the modules of the Orchestration and Choreography layer can exchange informational flows and services with other software systems **- Rule 3:** Only the Orchestration and Choreography layer can expose services for other software systems
Services	- Hosting the rules applicable to the software system informational entities - Implementation of the functions processing the software system informational entities	- Description of the state of the software system informational entities - Carrying out simple and complex controls - Data processing - Description of the services supported by the software system - Recording of status changes of the software system informational entities	- Computing rules - Information related to data processing: interest rates, legal information, …	**- Rule 1:** The Services layer guarantees the inter-business process consistency through checking the software system informational entities
Data Access	- Providing access to operational persistent data belonging to the software system by ensuring a real independence between processing and physical data models and performing data integrity controls	- Data selection - Data update - Data deleting - Data creation - Data edition - Table joining - Data integrity control	- List of tables to be used to store information related to an informational entity	**- Rule 1:** Only the modules of the Data Access layer provide data read and write services of operational persistent data belonging to the software system **- Rule 2:** Read and write services exposed by the Data Access layer are defined according to logical data model of the software system

4.2 Complementary information on the Orchestration and Choreography layer

We note that orchestration and choreography describe two complementary concepts related to processes execution. Orchestration describes how a central entity - called the coordinator - manages dependencies during the execution of services involved in a higher-level organizational process. Choreography focuses on the interactions between collaborating entities which may have their own internal orchestration processes. Such interactions are based on protocols that enable the conversation between the parties involved in choreography. According to [17], in choreography no organization necessarily controls the collaboration logic while orchestration is generally owned and operated by a single organization.

Orchestration and choreography are part of the organizational processes control, distributed across the ICP Integration area (Internal and external control applications) and the other information system applications whose orchestration and choreography layer is responsible for use cases monitoring. Each organizational process has a context containing temporary information including its status, newly created or modified information, and information already read. The process context is composed of two nested sub-contexts related to the information system and application levels. At the information system level, organizational processes contexts are managed by an application - called the Business Processes Management System (BPMS) - which belongs to the ICP Integration area, and updated based on information recorded by applications at the end of use cases. At the application level, the Orchestration and Choreography layer manages the sequence of tasks supported by the software system and the contexts of its use cases. Each use case is associated with a component - called use case driver - in the Orchestration and Choreography layer. Only use case drivers are allowed to access to use cases contexts. To implement the Orchestration and Choreography layer, a good practice consists in distinguishing two types of services: the transition services, and the request services. The former updates the use cases contexts while the latter's role is limited to reading these contexts. Apart from information already read and managed by the application, the content of the process context consists of information generated during the execution of this process, or collected from other applications. Therefore, information manipulated by a service managed in the 5+1 model Services layer are either persistent operational data accessed through the Data Access layer or temporary data provided by the context of the process with calls this service.

In addition to the management and internal control of the interactions between processes supported by an application, system, the Orchestration and Choreography layer manages the application interactions with other information system applications. Therefore, the layer is composed of three parts: Processes Internal Control (PIC), Management of Inbound Informational Flows (MIF), and Management of Outbound Informational Flows (MOF). Within the Orchestration and Choreography layer, the steering and control role is devoted to the PIC part while the MIF and MOF parts are informational flows converters and thus play a role similar to the Interface layer role.

4.3 Management of services in the 5+1 software architecture model

The 5+1 software architecture model manages software services according to many perspectives. Firstly, a service is either public or private. Secondly, a service is exposed by a software system either for end-users or for other software systems. Thirdly, a service is either used only by the information system applications or may be used by external information systems like partner's information systems. Finally, services may be viewed as integrating means of information systems. Therefore, the management of services in the 5+1 software architecture model is based on a typology which distinguishes five types of services: information system service, applicative service, end-user service, layer service, and component service. An information system service is a software service exposed by an information system for external information systems, and accessed via a specific application belonging to the Inbound and Outbound Flow Management Area of the Information City Plan (ICP). An applicative service is a software service exposed by an application for other applications belonging to the same information system, and accessed via the Orchestration and Choreography layer. An end-user service is a software service exposed by an application for human end-users, and accessed via the interface layer. Information system services and end-user services are usually composed of several applicative services. A layer service is a software service exposed by a layer for the other layers of a software

system. A component service is a software service exposed by a component for the components of the same software system layer. Table 2 explains when a service is public and when it is private.

4.4 The relationships between the 5+1 software model and the ICP areas

The software layers identified by the 5+1 software architecture model conform to the separation of concerns principle. Therefore software systems architected according to this model are scalable and adaptable to user needs. As a result, the importance of the layers of such applications depending on the ICP area hosting them can be taken into account during the development life cycle.

Table 2. Typology of services

Service	Public for	Private for
Information System service	- External Information Systems	- Information System applications - Human end-users
Applicative service	- Applications belonging to the same Information System	- External Information Systems - Human end-users
End-user service	- Human end-users	- Information System applications - External Information Systems
Layer service	- Other layers of the same software system	- Information System applications - External Information Systems - Human end-users
Component service	- Other components of the same layer	- Information System applications - External Information Systems - Human end-users - Other layers of the same software system

Table 3 provides an assessment of the importance of the layers of a software system implemented in a French insurance company and architected according to the 5+1 model. This company operates in many countries all over the world with more than 30,000 employees. It covers all insurance sectors including mass-risks such as motor car liability insurance or accident insurance, and industrial policies required by international companies. In recent years, it has reinforced its business in the personal insurance sector, income capacity and savings management, in particular through the promotion of pension-based life products. Lately, it has expanded its business from insurance to the wide area of asset management and financial services, and established a full-fledged bank structure in order to optimize the products and services. The current information system of this company is composed of more than a thousand applications running either on mainframes or on open systems. An urbanization project of this information system is ongoing to reach a target urbanized information system consistent with the ICP and the applicable architecture rules and principles. To assess the importance of the 5+1 software architecture model layers with respect to the ICP areas to which applications belong, we studied the architectures of many urbanized applications implemented according to the 5+1 software architecture model. We also collected additional information through interviews with the information system architects involved in the design and implementation of these applications. To complete Table 3, we have used a five-point Likert scale (0=Not important at all, 1=Weakly important, 2=Moderately important, 3=Important, 4=Very important). We note that the Technical Services layer is not included in Table 3 since it is important regardless of the ICP area. This table provides several indications. For example, it shows that the Interface and Navigation layers are very important for applications hosted by the Inbound and Outbound Flow Management Area while they are not relevant for applications belonging to the Shared Information area. Moreover, the Data Access layer is very important for applications hosted by the Business Intelligence area, the Shared information area, and the Business area.

Table 3. Importance of software layers depending on ICP areas

Areas	Software layers				
	Interface	**Navigation**	**Orchestration and Choreography**	**Services**	**Data Access**
Inbound and Outbound Flow Management	4	4	1	1	0
Party Relationships	1	1	3	3	3
Business Intelligence	3	0	1	1	4
Integration	1	1	4	1	1
Shared Information	0	0	2	3	4
Support	2	1	3	2	3
Business	1	1	3	4	4

Table 3 also shows that the most important layers are:

- Interface and Navigation for Inbound and Outbound Flow Management area applications;
- Orchestration and Choreography, Services, and Data Access for Party Relationships area applications;
- Interface and Data Access for Business Intelligence area applications;
- Orchestration and Choreography for Integration area applications;
- Services and Data Access for Shared Information area applications;
- Orchestration and Choreography and Data Access for Support area applications;
- Orchestration and Choreography, Services, and Data Access for Business area applications.

5. Conclusion and future research directions

In this paper, we have presented a software architecture model - called the 5+1 model - which helps build urbanized software systems. This model is compliant with information systems urbanization principles. First of all, the 5+1 model is organized in the same way than the Information City Plan (ICP) since it addresses the main urbanization principles used to define the ICP. On the one hand, the first urbanization principle is addressed by the 5+1 model which permits identifying - for each software system - a front-office composed of the Interface and the Navigation layers, and a back-office composed of the Orchestration, Services, and Data Access layers. On the other hand, the second urbanization principle is reflected by the 5+1 model which distinguishes four main processes supported by the back-office layers and two main processes supported by the front-office layers. Processes supported by the back-office layers are: Communication with the Information System applications, Tasks Management, Services Management, and Data Access Management. The Communication with the Information System applications and the Task Management processes are supported by the Orchestration and Choreography layer. The Services Management process and the Data Access processes are respectively supported by the Services and the Data Access layers. Processes supported by the front-office layers are: Management of the presentation of information and static aspects of interactions with end-users for each technical communication channel, and Management of the progress of screens kinematics. The former is supported by the Interface layer while the latter is supported by the Navigation layer. Moreover, the third urbanization principle is taken into account by the 5+1 model since the Technical services layer is shared by the front-office and the back-office while the Orchestration and Choreography layer allows front-office and back office to communicate.

Secondly, the 5+1 software architecture model contributes to the implementation of information systems urbanization in several ways. On the one hand, this model devotes a specific layer - the Orchestration and Choreography layer - to applications integration and software reuse. Indeed, software services exposed by information system applications for reuse facilitate the integration of the applications using them. On the other hand, the 5+1 model contributes to the information system agility. As stressed previously, the 5+1 model conform to the separation of concerns principle. Therefore, a software system architected according to this architecture model is scalable and adaptable to users' needs. In particular, the importance of the layers of such applications depending on the ICP area hosting them can be taken into account during the development life cycle. As a result, computerization resources can be allocated efficiently according to the importance of the layers of the software system under development. However, this model should be evaluated through experimentation in order to better use it in practice. Furthermore, two questions remain unanswered. The first question concerns the effectiveness of using the 5+1 model for architecting a Decision Support Systems and the second is related to the integration of software systems architected using this model with enterprise systems like ERP and CRM systems. These issues are two future research directions.

References

[1] F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, vol. *20*, no. *4*, pp. *10-19*, April, 1987.

[2] F. P. Brooks, Jr., *The Mythical Man-Month*, Boston, MA, USA: Addison-Wesley Longman Inc., 1995.

[3] D. L. Parnas, "Software Aging," in *Proceedings of the 16th International Conference on Software Engineering* (*ICSE'94*), IEEE Computer Society/ACM Press, Sorrento, Italy, May 16-21, 1994, pp. *279-287*.

[4] S. Guetat and S. B. D. Dakhli, "The Information City: A Framework for Information Systems Governance," in *Proceedings of the 5th Mediterranean Conference on Information Systems* (*MCIS'2009*), AIS Digital Library, 2009.

[5] C. Toffolon, "L'Incidence du Prototypage dans une Démarche d'Informatisation," PhD. Dissertation, Université de Paris-IX Dauphine, Paris, France, 1996.

[6] C. Toffolon and S. Dakhli, "The Software Engineering Global Model," in *Proceedings of the COMPSAC'2002 Conference*, IEEE Computer Society Press, Oxford, United Kingdom, August 26-28, 2002, pp. *47-53*.

[7] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practice*, 2nd edition, Reading, MA, USA: Addison-Wesley, 2003.

[8] G. Booch, "The irrelevance of architecture," *IEEE Software*, vol. *24*, no. *3*, pp. *10-11*, 2007.

[9] R. N. Taylor, N. Medvidovic and E. Dashofy, Software *Architecture: Foundations, Theory, and Practice*, New York, NY, USA: Wiley Publishing, 2009.

[10] M. Shaw and D. Garlan, *Software Architecture: Perspectives on a Emerging Discipline*, Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[11] P. Kruchten, H. Obbink and J. Stafford, "The past, present and future for software architecture," *IEEE Software*, vol. *23*, no. *2*, pp. *2-10*, 2006.

[12] P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd edition, Reading, MA, USA: Addison-Wesley Professional, 2003.

[13] Software Engineering Standards Committee of the IEEE Computer Society. (2000, September, 21). *IEEE Recommended practice for architecture description of software-intensive systems, IEEE Std 1471-2000*, IEEE-SA Standards Board [Online]. Available: http://standards.ieee.org/

[14] A. Jansen, J. Bosch and P. Avgeriou, "Documenting after the fact: Recovering architectural design decisions," *Journal of Systems and Software*, vol. *81*, no. *4*, pp. *536-557*, 2008.

[15] D. E. Perry and A. L. Wolf, "Foundations for the Study of Software Architecture. Software Engineering Notes," *ACM SIGSOFT*, vol. *17*, no. *4*, pp. *40-42*, 1992.

[16] J. Coutaz, "Architectural design for user interfaces," in *Encyclopaedia of Software Engineering*, J. J. Marciniak, Ed., New York, NY, USA: John Wiley & Sons, 1994, pp. *38-49*.

[17] T. Erl, *Service-oriented architecture: concepts, technology, and design*, Upper Saddle River, NJ, USA: Prentice-Hall International, 2005.

[18] C. Toffolon, "The Software Dimensions Theory," in *Enterprise Information Systems, Selected Papers Book*, Joaquim Filipe, Ed., Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[19] P. P. Lemberger, *Managing the Complexity of Information Systems: The Value of Simplicity*, New York, NY: Wiley & Sons, 2011.

[20] M. M. Lehman and L. A. Belady, Eds., *Program evolution: processes of software change*, Waltham, MA, USA: Academic Press, 1985.

[21] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *IEEE Proceedings*, vol. *68*, no. *9*, pp. *1060-1076*, 1980.

[22] G. Lakoff and M. Johnson, *Metaphors we live by*, 2nd Ed. Chicago, IL, USA: University of Chicago Press, 2003.

[23] D. Dieberger and U. F. Andrew, "A City Metaphor to Support Navigation in Complex Information Spaces," *Journal of Visual Languages and Computing*, vol. *9*, no. *6*, pp. *597-622*, 1998.

[24] C. Knight, "System and Software Visualization," in *Handbook of software engineering & knowledge engineering. (vol. 2): Emerging technologies*, Change S. K., Ed., River Edge, NJ, USA: World Scientific Publishing Company, 2002, pp. *131-139*.

[25] J. Ross, "Creating a strategic IT architecture competency: Learning in stages," *MISQ Quarterly Executive*, vol. *2*, no. *1*, pp. *31-43*, 2003.

[26] D. Garlan and M. Shaw, "An introduction to software architecture," in *Advances in Software Engineering and Knowledge Engineering*, vol. *2*, pp. *1–39*, Hackensack, NJ, USA: World Scientific Publishing Company, 1993.

[27] A. Jansen, "Software architecture as a set of architectural design decisions," in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 2005)*, IEEE Computer Society Press, Pittsburgh, USA, 2005.

[28] P. B. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, vol. *12*, no. *6*, pp. *42-50*, 1995.

[29] C. Hofmeister, R. Nord and D. Soni, *Applied Software Architecture*, Reading, MA, USA: Addison-Wesley Publishing, 1999.

[30] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord and J. Stafford, *Documenting Software Architectures:Views and Beyond*, Reading, MA, USA: Addison-Wesley Publishing, 2002.

[31] P. Clements and L. Northrop, *Software Product Lines, Practices and Patterns*, Reading, MA, USA: Addison-Wesley Publishing, 2002.

[32] J. Bosch, *Design & Use of Software Architectures: Adopting and evolving a product-line approach*, Reading, MA, USA: Addison-Wesley Publishing, 2000.

[33] A. Hussey and D. Carrington, "Comparing the MVC and PAC architectures: a formal perspective," *IEEE Proceedings on Software Engineering*, vol. *144*, no. *4*, pp. *224-236*, 1997.

[34] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80," *JOOP*, vol. *1*, no. *3*, pp. *26-49*, 1988.

Biographical notes

Sana Guetat
Sana Guetat received the Master's degree in Accounting from the University of Manouba, Tunisia and the PhD. degree in Management Science from the Toulouse I University, Toulouse, France.
She is currently an Assistant Professor of Management Science, and Information Systems at the Maine University, Le Mans, France, where she is a member of the ARGUMANS Laboratory and the head of the Master of Business Administration. Her research interests include Accounting, Finance, Audit, Information Systems Architecture, and Knowledge Management.

www.shortbio.net/sana.guetat@univ-lemans.fr

Salem Ben Dhaou Dakhli
Salem Ben Dhaou Dakhli received a Statistician Economist Diploma from the National School of Statistics and Economic Administration (ENSAE), Paris, France, the Master's degree in Applied Mathematics, the Master's degree in Computer Science, and the PhD. degree in Computer Science from the Paris-Dauphine University, Paris, France. He is currently a Professor of Statistics, decision theory, software engineering, and Information Systems Architecture and a member of the CERIA Laboratory at the Paris-Dauphine University. His research interests include Software Engineering, Information Systems Architecture, Software Economics, and Knowledge Management. Dr. Dakhli is a TOGAF Enterprise Architect certified by the Open Group and a consultant in Information Systems with many French Banks and Insurance Companies.

www.shortbio.net/sdakhli@computer.org