Architecture Views illustrating the Service Automation Aspect of SOA

Qing Gu[1], Félix Cuadrado[2], Patricia Lago[1], and Juan C. Dueñas[2]

[1] Dept. of Computer Science
VU University Amsterdam, The Netherlands
[2] Dept. de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid, Madrid, Spain

Abstract. Service-Oriented architecture (SOA) as an emerging architecture style has been widely adopted in the industry. Due to the heterogeneous, dynamic and open nature of services, architecting Service-Based Applications (SBAs) poses additional concerns as compared to traditional software applications. Hence, for SOA architects it is of great importance that their concerns are appropriately addressed in the architecture description. However, an effective and systematic way of documenting SOA design is currently missing. In this work, we focus on the service automation aspect of SOA. We carried out two large case studies to learn the industrial needs in illustrating services deployment and configuration automation, from now on service automation. As a result, we broke down service automation into three important sub-aspects, and we developed a corresponding set of architecture views (automation decision view, degree of service automation view and service automation related data view) that expresses the different concerns of stakeholders who share interest in service automation. This set of views adds to the more traditional notations like UML, the visual power of attracting the attention of their users to the addressed concerns, and assist them in their work. This is especially crucial in service oriented architecting where service automation is highly demanded.

Key words: Service-oriented Architecture;Service-based Application;Architecture view;Automation;Service management;Service deployment

1 Introduction

Service-oriented architecture (SOA) as an architectural style has drawn the attention from both industry and academia. SOA-based systems (i.e., Service-Based Applications or SBA) are constructed by integrating heterogeneous services that are developed using various programming languages and running on different operating systems from a range of service providers. Services are loosely coupled entities, often designed under open-world assumptions, distributed across organizational boundaries and executed remotely at their service providers' environment. They require a smoother transition from development to operation than traditional applications.

Consequently, the architecting of SBAs pose additional concerns as compared to traditional software applications. Some examples of these concerns include how to reason about a SOA design and how to represent the characteristics of SOA that the design delivers, how to architect the SBA to operate in an unknown environment, or how business processes can be supported by means of the collaboration of multiple services.

Clearly, traditional software engineering and architecting techniques, methods and tools are no longer sufficient to deliver SBAs, as they do not take into account specificities of services, such as the need for smooth transition from development to operation, the need to integrate third-party components, or the possibility to be hosted by a different organization. Therefore, it is necessary to propose new techniques that supplement the traditional models, enabling the capture at design time of all the relevant information about those new concerns and improving the usefulness of the architecture description.

We have chosen as the aspect under study the degree of automation of SBAs. The current situation is that the design decisions on whether a service can be possibility automated, its benefits and limitations, or the degree of automation are often left implicit in the architectural design and its description. Our goal in this paper is to make them explicit and to find a notation useful for this purpose, able to be understood for most stakeholders (including the user if relevant to the domain).

We carried out two large case studies to learn the industrial needs in illustrating services deployment and configuration automation, from now on service automation. As a result, we broke down service automation into three important sub-aspects, and we developed a corresponding set of architecture views (automation decision view, degree of service automation view and service automation related data view) that expresses the different concerns of stakeholders who share interest in service automation.

The first one (the decision view) conveys the decisions about service automation by making explicit which architecture constraints may impact the degrees of automation, and which services are affected by each constraint. The degree view -second one- shows the degree of service automation that the service flow is expected to achieve, but not the details on how to get it. Last, the automation-related data view contains explicit information about the generation, management and provision of additional input that are required from either human actors or policies.

In addition to constructing these views, we highlighted the added-value of the graphic notations we used. We argue that this set of views adds to the more traditional notations like UML, the visual power of attracting the attention of their users to the addressed concerns, and assist them in their work. Moreover, we also reflected on the relationship between the degree of automation and the granularity of services and the applicability of these views to SOA in general.

The reminder of the chapter is organized as follows. In Sec. 2, we provide some background information on architecture views and management systems for SBAs. In Sec. 3, we discuss the need of documenting SOA design decisions and rationale in effective illustrations and present a set of concerns that we elicited from the case studies, which points out what needs to be illustrated in SOA architecture description. With these requirements in mind, we present the three service automation views in Sec. 4, 5 and 6 respectively. We highlight the power of visualization in Sec. 7 and we discuss our observations in Sec. 8. We conclude the chapter in Sec. 9.

2 Background information

2.1 Architecture views

The architecture of a software system should be documented with the purpose of capturing early design decisions, providing re-useable abstractions of software systems and enabling communication of the software architecture among stakeholders [1]. To produce relevant documentation for a software system, one has to decide what information needs to be documented and which notations or diagrams are suitable for representing this information. These decisions heavily depend on who is the target reader of the documentation.

A software system typically involves multiple *stakeholders* that have different concerns. For instance, the architect is concerned about the structure of the system; the project manager is concerned about the resources (e.g., cost, time, number of developers) needed for developing the system; and the developer has concerns about the implementation of the system. The architectural design of the system therefore should be documented in such a way that the concerns of each stakeholder are addressed.

Following the *separation of concerns* principle, software architects have already been using multiple views for years to represent the software systems of interest from multiple perspectives. These views facilitate the management of the complexity of software engineering artifacts. For instance, a structure view can be used to describe the construction of a software system (including e.g. components and connectors); while a data view can be used to describe the data flow between the components. By representing the architecture of the system in these two separate views, the software architect may focus on the construction design of the system by using the structure view, while the data manager may concentrate on the management of data by using the data view.

One of the original goals behind IEEE 1471 and ISO/IEC 42010 was to "establish a frame of reference of terms and concepts for architectural description" [2]. This frame of reference provides a basis for the community to develop a collection of views which addresses concerns that occur commonly across software projects. Practitioners may directly benefit from the application of these viewpoints in that they enable an effective architecture description.

However, the existing reusable views are limited in the sense that they address concerns that often appear in traditional software architectures. With the wide adoption of recently emerged software architecture styles (like SOA), additional concerns (often specific to the architecture styles) challenge the reusability of the existing viewpoints. The lack of available views make practitioners face difficulties to find an effective way to illustrate any new characteristics introduced by any architecture style. As a result, views that enable the illustration of specific concerns introduced by modern software architecture styles are needed.

2.2 Management system for SBAs

Hegering [3] described the management of networked systems as the set of measures necessary to ensure the effective and efficient operation of a system and its resources, according to an organization's goals. In service-based applications the functionality is not provided by individual, monolithic elements, but is achieved by collaboration between multiple services. In order to get this collaboration, the management system must be aware of the participating elements, deploy and configure them if necessary. This is a complex process, because as the number of services grows, the possible combinations that must be considered by the management system increase exponentially. On top of that, the distribution of the participating elements over a computing network further complicates the process. Those are common characteristics for every SBA. However, they are not the only relevant factors. The domain-specific characteristics of each SBA, such as the characteristics of the services, the capabilities of the runtime resources, or the organization's business aspects must also be supported by the management system. It is clear that the potential variation of all the factors complicates defining a general solution for SBAs deployment and configuration.

In the field of systems management, there are two opposite approaches for controlling the deployment and configuration process: traditional management processes and autonomic management [4]. In traditional processes, a human administrator is continuously in control of the change process. He / She diagnoses the system, manually defines the required changes and controls every aspect of the execution. This approach is very costly and cumbersome, because it implies that every activity executed by the architecture must be performed or at least validated by a human actor. On the other hand, autonomic computing promotes to automate as much as possible the operation of the system. Ideally, completely automated closed control loops are implemented, where the system reacts automatically to a change in the environment, diagnoses its severity and implications and applies the required corrections in order to restore the environment functionality. This approach eliminates the bottleneck inherent to human operation, consequently improving scalability and efficiency of the management system.

Although autonomic control would be the most desirable approach, it is not always feasible to achieve it, because of either technical factors (e.g., a monitoring interface from a managed server does not provide information about service faults so a human administrator has to manually diagnose the incidences by inspecting the server and system logs) or, organizational aspects (e.g., manual control is preferred because the service update process is considered critical for the organization, so an automated system cannot have complete control over the process). For most cases an adequate balance between traditional and autonomic management will be the right approach. Management systems should pursue the autonomic approach to the greatest extent possible, while respecting the requirements derived from the domain of application.

Supporting the diversity of managed services, operation environments and organizational aspects with the same management architecture demands a high level of flexibility, which is pushed forward adopting a service-oriented approach. Service orientation can offer great flexibility and agility so that the architecture can easily adapt to the characteristics of different environments with reduced required configuration. The Service Deployment and Configuration Architecture (from this point onwards SDCA) [5] is an example of such an approach, which is further described in Sec. 3.1.

3 The requirements for illustrating the automation aspect of SBAs

The SOA paradigm promotes creating new functionality from the dynamic combination of services provided by different stakeholders. A SBA can be viewed as a set of dynamically combined services.

While automation in a traditional software system refers to the degree to which the execution of the *software process* can be automated without human intervention, automation in SOA systems refers to the degree to which *services*, comprising the system of interest, can be executed automatically without any human intervention. While the two definitions are quite similar, due to a set of characteristics that differentiate SBAs from traditional software systems [6], in service-oriented development the decision on the degree of automation of each service is heavily influenced by (and has impact on) at least two quality attributes.

The first quality attribute is trust, i.e. confidence (especially from the users perspective) on the truth of what delivered or promised. SBAs are typically not fully controlled by the company: some integrated services execute in the domain of remote, dynamically determined service providers, and can be discovered and integrated at runtime. This means that if something goes wrong, malfunctions might decrease the satisfaction of ones customers, and hence influence the overall company business. Especially in traditional business domains (like banking and services to the public) the tendency is to develop applications with service-oriented technologies, but with the properties of old fashion software systems: low level of automation, static integration of services, no dynamic discovery and no dynamic composition. In the case of required interaction with third-party services, the requirements -both functional as non functional- of these and the penalties for failure, are governed by Business Level Agreements (BLA) or Services Level Agreements (SLA).

The second quality attributes is reliability, i.e. the ability of a software system to perform its required functions under stated conditions for a specified period of time [7]. By automatically integrating services during execution, reliability of the whole execution depends on various unpredictable factors, like the correct specification of the requirements of the services to be dynamically integrated, availability of such services, or their correct execution. If third-party service discovery & composition is automated, the company does not have anymore full control on the software products delivered to its customers.

It is often claimed that SBAs have the agility to adapt to customer needs by automatically reacting to continuous changes in business processes. Consequently, the more services in a SOA system can be automated (i.e., do not need humans to make decisions for execution), the higher agility a SBA can achieve.

Users of highly automated SBAs clearly benefit from less human intervention and thus less labor costs. However, automating the execution of services and delivering agile and reliable SBAs is not always possible (as we explained above in the examples of trust and reliability) and poses additional concerns. Some of the concerns relate to the decisions on the degree of service automation; while some of the concerns are related to the realization of these decisions.

However, design decisions and their associated rationale on whether a service can be possibility automated, the benefits and limitations of automating a service, or how to automate a service are often either ignored or left implicit in the architectural design and its description. In spite of the evidence for the need of documenting design decisions and rationale in effective illustrations [8, 9] little work exists so far in the area of SOA [6]. This need has been further highlighted in the S-Cube analysis of the state of the art in [10], where one major challenge is in identifying and representing relevant concerns in SBA engineering, like monitoring, self-organization and adaptation. Viewpoints are mentioned as means to capture multiple perspectives on a given SBA. Though, they are meant to aid engineering of specific systems, whereas the corresponding architecture descriptions have not been sufficiently addressed yet. This motivated us to investigate what the stakeholders are concerned about with respect to service automation and how to address these concerns in the architecture description.

To answer this question, we analyzed the service automation aspect of the SDCA as well as two concrete industrial case studies where the SCDA has been applied to. The first case study (BankFutura) describes the deployment and configuration system of a banking organization. As for most enterprise systems, in this case the non-functional requirements such as the criticality of the delivered services, guaranteed performance levels, and organizational aspects are the dominating factors for driving the decisions on the degree of automation of the service execution flow. In the second case (HomeFutura) the implementation of SDCA provides the services of multiple third-party providers, which are presented to the end users through a service catalog, allowing them to select the functionality they require. While the same service execution flow is adopted in both cases, there are significant variations in the automation related aspects, due to the impact of their domain-specific and organization-specific constraints.

In the remaining of this section, we present an overview of the SDCA and its industrial case studies (BankFutura and HomeFutura), focusing on the concerns related to service automation that we have elicited from the cases. These concerns serve as the requirements for illustrating the automation aspect of SOA in the architecture description.

3.1 The Service Deployment and Configuration Architecture

The Service Deployment and Configuration Architecture (SDCA) is a flexible, service-oriented management architecture that can address the requirements of distributed, heterogeneous SBAs (a.o. dynamic discovery, dynamic composition, adaptation, runtime evolution). The management functions are provided by a set of services, which collaborate to identify the required changes to the environment in order to fulfill the SBA business objectives (a.o. service availability). SDCA Services are automated, reasoning over models representing the characteristics of the managed services and the runtime environment. Finally, in order to adapt to the domain-specific characteristics, the specific behavior of the services can be customized through the definition of policies that govern the decisions taken over the process.

The objective of the SCDA is to provision new functionality (in the form of services) by identifying and applying a set of changes to the managed environment. This function is achieved by an execution flow consisting of the combined invocation of nine deployment services, as shown in Fig.1.

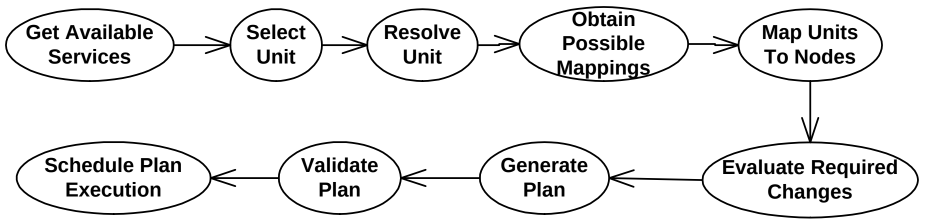


Fig. 1. The SDCA deployment service execution flow

A typical execution starts when an external change to the system is triggered (e.g. an updated version of a service has been released and must be deployed, or a hardware malfunction caused a server to stop working, and the affected services must be redeployed at another node). After it is decided that a change is necessary, the deployment service Get Available Units is invoked. This first step retrieves the complete list of units and services currently available. The second step is the deployment service Select Unit, where one of those available units is selected, in order to be deployed to the environment. The *unit selection criteria* will be provided to the service as an external input. After that, the deployment service Resolve Unit is invoked, where the deployment unit containing the service is analyzed, in order to find a closed set of units satisfying all their dependencies. There might be multiple candidate units satisfying one dependency (e.g. multiple units with minor, compatible versions) and for those cases a *criteria for selecting among them* must be provided as external input. Once the complete set of units that will participate in the operation has been identified, the deployment service Obtain Possible Mappings evaluates the available resources from the container, and returns for each unit the potential nodes of the environment where those can be deployed. Starting with that input, the deployment service Map Units To Nodes decides on the final destination for each one of the involved units, according to external *distribution criteria*. After those mappings have been established, the deployment service Evaluate Required Changes compares the current environment status with the desired changes in order to obtain the set of required changes that must be applied to it (e.g. install selected deployment units if they are not currently running at the environment). Those changes are packed and sorted into a deployment plan in the deployment service Generate Plan, whose purpose is to ensure a correct execution of the list of changes, by adding restrictions to their execution order. Defined plans can either be instantly applied to change the environment, or can be temporarily stored at a change repository. Before being applied to the environment, plans must pass through the deployment service Validate Plan. This step checks that the automatically obtained plan is coherent with the environment state, and will obtain the desired result. Finally, the deployment service Schedule Plan Execution receives the accepted plan and schedules it for execution at some point in the future, which will be also determined by an external *schedule agenda*.

Some of the deployment services can be *completely automated* as they do not require any external intervention, such as Get Available Service and Obtain Possible Mappings; whereas some others cannot be completely automated as *external input* (i.e., additional input external to the service invocation flow)(e.g., distribution criteria, unit selection criteria) are required during the deployment process. The architect of SDCA provided services requiring external input with certain degree of flexibility, supporting two alternatives for their implementation. One of the solutions is to create a user interface so that a dedicated human actor can provide the required input to those services. Another solution is to formalize the necessary knowledge for providing the required input in terms of policies, which can be automatically consumed bt the deployment service. SDCA services with the format approach are called *semi-automated services* whereas the latter are called *policy-driven automated services*.

The stakeholders of the SDCA that are concerned about service automation include *the SOA architect*, who is responsible for defining a deployment service flow that can support the automated provisioning of services, adapting to the hardware characteristics of each environment; *the SOA manager*, who governs the design and implementation of the SDCA, and *the users of the SDCA*, who are often the SOA architects that intend to apply the SDCA to specific domains.

Being the designer of the SDCA, the SOA architect is mainly concerned about how to provide enough flexibility with the degree of automation, in order to allow adaptation of the flow to specific domain requirements. Additionally, the SOA architect is concerned about how to support the other stakeholders in terms of service automation. SDCA users are mainly concerned about how to customize the SDCA in such a way that domain specific constraints are fulfilled.

The complete list of concerns of each stakeholder is presented in Tab. 1, where each concern is described by its associated stakeholder, a description, and a concern ID.

Table 1. Concerns relevant to service automation in the SDCA

Stakeholder	Concern ID	Concern description
SOA architect	SDCACon1	Justify whether their decisions on the degree of service automation are reasonable.
	SDCACon2	Provide enough flexibility with the degree of automation, in order to allow adaptation of the deployment service flow to specific domain requirements.
	SDCACon3	Suggest policies that are required for assisting the deployment service flow.
SOA manager	SDCACon4	Trace, verify and control the decisions on the degree of service automation.
	SDCACon5	Gain an overview of the degree of service automation supported by the SDCA.
	SDCACon6	Gain an overview of the re-configurability of the SDCA.
SDCA Users	SDCACon7	Be aware of which deployment services are domain specific (hence customization is needed) and which ones are domain independent (hence no customization is necessary).

3.2 BankFutura: An application of the SDCA to an enterprise domaina

A Spanish banking company, called from this point on BankFutura, with several millions of clients over the world, and more than two thousand branches, renovates its services portfolio, which includes client services (internet banking, cashiers), internal services (for company workers at the bank offices) and B2B services for inter-bank transactions. As those services capture the company knowledge, they are internally developed and provided, with no third party dependencies. This is understandable, as they constitute the core of the company business and consequently must be under full control of the company. The company services have been architected following the SOA / BPM paradigm, in order to cope with the complexity.

The services runtime infrastructure that will replace the legacy systems and mainframes is composed by artifacts such as relational databases, JEE application servers, and BRM (Business Rule Managers) systems. Each artifact of the system is presented as a banking service, hiding its implementation details and providing a uniform high-level view. Banking services are published in directories and connected through an ESB (Enterprise Service Bus). The complete runtime infrastructure is dimensioned and defined beforehand, in order to support the strict non-functional requirements for the service operation, as well as adequately support the types of services that will provide the core banking functionality.

In BankFutura, every deployed banking service must be always available, respecting the requirements defined at the SLA, while dispatching the requests from a potentially enormous number of consumers. Neither hardware and software malfunction, or denial of service attacks from ill-intended actors, should be able to disrupt the service operation, as service downtime would imply huge monetary costs. The stability of the banking system becomes one of the most important non-functional requirements.

Another non-functional requirement for the banking system is security. The exchanged information of banking services is very sensible, as it contains the financial status and personal data of the clients, so it is not only critical for their trust but also legally protected by the personal data confidentiality regulations. Because of that, it is fundamental to safeguard the security of the underlying systems, and provide complete logging and traceability of the performed operations. This way, change initiation, approval and execution must be registered and supported by the change management architecture, including a responsibility chain for any identified incidences.

In order to respect all these restrictions and facilitate at the same time system evolution, the BankFutura infrastructure is replicated into several, tiered environments (integration, pre-production and production) which present a balance between agility of changes and criticality. The complexity and pre-defined structure of the deployment and configuration architecture justifies that BankFutura employs specialized staff, such as Environment administrators, for watching over the runtime health, diagnosing malfunctions and controlling the execution of planned changes to the environment, despite the costs the bank is incurring by keeping this staff.

The stakeholders who are concerned about service automation in BankFutura include the SOA architect, the banking deployment plan creator, the environment administrator, and the service deployment manager.

The SOA architect is responsible for applying and customizing the SDCA so that the resulting deployment service flow can support the complete provisioning of the released services to the several tiered environments of the infrastructure of the company. He/She is mainly concerned about automating the deployment services as much as possible (moving away from handcrafted scripts) while at the same time integrating human control and responsibility over the complete process.

The banking deployment plan creator and *the environment administrator* are both deployment actors in the banking deployment service flow. The former is responsible for creating a deployment plan which will provide the desired functionality when applied to the environment; while the latter is responsible for the correct configuration of the managed infrastructure and the selection of the right physical node for each newly deployed service, taking into account the additional resources consumption by each new service. Both of them are mainly concerned about how to perform their roles in the banking deployment service flow.

The deployment manager is in charge of supervising the execution of banking deployment service flow and ensuring that the deployed banking system is aligned with the business objectives of BankFutura.

The detailed concerns of each stakeholder are listed in Tab. 2, where each concern is presented with its associated stakeholder, a description, and a concern ID.

Table 2. Concerns relevant to service automation in BankFutura

Stakeholder	Concern ID	Concern description
SOA architect	BankCon1	Justify whether their decisions on the degree of service automation are reasonable given the specific constraints in the BankFutura.
	BankCon2	Understand what specific constraints affect each deployment services and how each constraint influences the degree of service automation.
	BankCon3	Analyze the possible alternatives on the degree of service automation in order to evaluate how the deployment service flow can react to changing requirements or constraints.
The deployment manager	BankCon4	Trace, verify and control the decisions on the degree of service automation.
	BankCon5	Gain an overview of the degree of service automation in the BankFutura deployment service flow.
	BankCon6	Ensure that the environment administrator and banking deployment plan creator carry out the assigned tasks as expected and are able to trace responsibility in case an error occurs.
	BankCon7	Ensure the availability of required policies that are required for the deployment services in time.
	BankCon8	Ensure that the required policies for the deployment process are aligned with the organizational goals and regulations.
Environment administrator	BankCon9	Ensure the stability of the managed environment after executing the deployment services.
	BankCon10	Define the role and responsibility in preparing policies.
	BankCon11	Define the role and responsibility in the deployment service flow.
Banking deployment plan creator	BankCon12	Select the right physical node for each newly deployed service, taking into account the reasons that led to the initial definition of the environment topology
	BankCon13	Know which services (and what version of the service) must be made available in each environment.
	BankCon14	Define the role and responsibility in the deployment process.

3.3 HomeFutura - An application of the SDCA to a personal domain

The service aggregator of this case study (called from this point on HomeFutura) wants to offer subscribers a large catalog of services that can be consumed from the devices available at the digital home. The digital home is the house of the near future, an always connected entity, provided with network and devices to access Internet resources. It allows users to consume a wide range of services; multimedia entertainment services, surveillance services, e-learning services or consumer electronics control, just to mention a few. Services are provisioned over the Internet and accessed through multiple home devices. The specific hardware elements that will be available are controlled by the end users, which can dynamically decide to acquire additional equipment.

The ultimate goal is to create an environment that benefits end users, service providers, and service aggregators. End users should be able to browse all available services and subscribe to those they are interested in, automatically accessing them without technical skills. Service providers develop and offer services to be consumed by the end users. Service aggregators are the point of contact with the users, managing their subscription, interacting with the service providers, and ensuring correct and seamless service provisioning.

In this case study the deployment architecture plays a fundamental role. In order for those services to be available, it is necessary to execute deployment and configuration activities over the home infrastructure. The general characteristics of the environment are similar to the previous case, with the required operations consisting of managing services running over a distributed, heterogeneous infrastructure. However, the specific characteristics of this scenario lead to a different solution. In contrast with the case study of BankFutura, environment stability is not the dominating constraint. This is because the domain is personal, and the services are consumed and used without warrantee of performance nor agreed Quality of Service expressed through BLAs or SLAs. On top of that, guaranteeing stability is much harder, because of the high degree of uncertainty about the specific equipment that will be available at every moment.

Instead, the fundamental goal in HomeFutura is being able to provide the end user with a seamless experience in the process of acquiring new services. The user is not concerned about the technical details behind the services or the installation process. Those aspects must be correctly managed by the architecture, while the user is only informed about the relevant information, like functionality, or pricing.

The stakeholders who are concerned about service automation in HomeFutura include the SOA architect, the service aggregator, and the end user.

The SOA architect is responsible for defining the architecture of the digital home service deployment system by applying the SDCA. The main concern consists of how to provide a flexible deployment system that is able to adapt to the available infrastructure at each home while at the same time hiding all the technical details from the end user.

The role of a *service aggregator* is to manage the service catalog available to the different users and handle the signed contracts with service providers, ensuring that the portfolio of services offered to the users can have all their technical dependencies correctly satisfied. The service aggregator is also responsible for providing selection policies which determine what providers / versions for the services can be accessed by each different client.

The end user consumes the available services offered by HomeFutura, demanding as much variety in the services catalog as possible. The end user is mainly concerned about the simplicity of the process of accessing the desired functionality.

The detailed concerns of each stakeholder are listed in Tab. 3, where each concern is presented with its associated stakeholder, a description, and a concern ID.

Table 3. Concerns relevant to service automation in HomeFutura

Stakeholder	Concern ID	Concern description
SOA architect	HomeCon1	Justify whether the decisions on the degree of service automation are reasonable given the specific constraints in HomeFutura.
	HomeCon2	Understand what specific constraints affect each deployment service and how each constraint influences the degree of service automation.
	HomeCon3	Analyze the possible alternatives on the degree of service automation in order to evaluate how the deployment service flow can react to changing requirements or constraints.
	HomeCon4	Design a highly automated deployment process, with a minimal requirement on human intervention
The deployment manager	HomeCon5	Trace, verify and control the decisions on the degree of service automation
	HomeCon6	Gain an overview of the degree of service automation in the HomeFutura deployment service flow.
	HomeCon7	Ensure the policies that are required in the deployment process are ready in time
	HomeCon8	Ensure the policies that are required in the deployment process are aligned with the organizational goals and regulations
Service aggregator	HomeCon9	Define the role and responsibility in preparing policies.
End user	HomeCon10	Participate in the deployment process the simplest way possible

3.4 Summary

From the analysis of the SDCA and two industrial case studies, we observed that service automation is especially important during design and is relevant to multiple stakeholders in that we identified a considerable number of service automation related concerns. Being considered, designed and implemented, however, those concerns have not been explicitly addressed in the architecture description.

Instead, the current architecture description of the SDCA (as well as the two case studies) addresses the service automation related concerns in a very abstract way. For instance, it is stated that the SDCA provides a flexible solution that can be easily customized in various domain applications. However, the information about how flexible the solution is, how easy the solution can be applied, and how to customize the SDCA in specific domains is lacking. Hence, there is a need to find an effective way to illustrate how the concerns related to service automation are addressed in the architecture description.

In other words, we face the questions of *what information should be documented in the architecture description* and *how to document it in an effective way so that the stakeholders can easily understand it.* To answer the first question, we synthesized the concerns listed in Tab. 1, Tab. 2, and Tab. 3. The reason for doing so is that we noticed that a reasonable numbers of concerns are overlapping and demanding for the same type information. For instance, the concerns with ID SDCACon1, BankCon1 and HomeCon1 are all about justifying the decisions on service automation but in different cases (hence overlapping); and concerns with ID SDCACon3, BankCon7, BankCon10, BankCon13, HomeCon7, Home-

Con9 all demand for illustrating the information that is related to generate and access policies.

After the synthesis, we identified eight main concerns that are representative for the complete set elicited from the SDCA and its two case studies. *Decision on the degree of automation* covers all the concerns that are related to the decisions on service automation and their justification ; *Reconfigurability in terms of automation* covers all the concerns related to alternatives on the degree of service automation; *The impact of architecture constraints on the degree of automation* covers all the concerns related to domain-specific constraints; *Degree of automation* covers all the concerns related to the degree of automation a SBA can achieve; *Accountability* covers all the concerns related to the responsibility of stakeholders; *The preparation of policies* covers all the concerns related to the readiness of policies; *The specification of policies* covers all the concerns related to the content of policies; and *Human participation* covers all the concerns related to human actors with regards to their involvement in the deployment service flow.

Further, we noticed that some of these main concerns are inter-related. More specifically, the first three main concerns are about decisions, alternatives and constraints, which form a cause-effect-rationale relation. As such, we decided to address all these concerns using a **decision view**. The next two main concerns are about the degree of automation resulted from the design and the impact of such a degree on the execution of the deployment service flow. As such, we decided to address these concerns using a **degree view**. The last three main concerns are about the policies and human participation for enabling different degrees of service automation. Since policies and input from human actors can both be considered as data, we decided to use a **data view** to address the concerns. Hence the automation decision view, degree of service automation view and service automation related data view illustrate the service automation aspect for the architecting of SDCA.

The mapping between the elicited concerns, synthesized concerns and views for addressing these concern is presented in Tab. 4.

4 The automation decision view

The automation decision view is designed to illustrate all the decisions that have been made on the degree of service automation, the rationale behind them, and the impact of domain specific constraints on the decisions. With these requirements in mind, we created a set of graphic notations for constructing the automation decision view, as no other notation in the literature fits to our purposes. These graphic notations are presented in Fig. 2.

In this figure, services are represented by ovals; the three ones in the first column represent the three different degrees of service automation that have been decided. Moreover, they also indicate that alternative degrees of service automation are not feasible or reasonable. The services in the second column represent a decision has been made or left open among alternative degrees of

Table 4. Mapping between the elicited concerns, synthesized concerns and views

View	Main concern	Concerns in the SDCA	Concerns in BankFutura	Concerns in HomeFutura
Automation decision view	Decision on the degree of automation	SDCACon1, SDCACon4	BankCon1, BankCon4	HomeCon1, HomeCon5
	Reconfigurability in terms of automation	SDCACon2, SDCACon6	BankCon3	HomeCon3
	Impact of architecture constraints on the degree of automation	SDCACon7	BankCon2	HomeCon2
Degree of service automation view	Degree of automation	SDCACon5	BankCon5	HomeCon4, HomeCon6
	Accountability	-	BankCon6, BankCon11, BankCon14	HomeCon10
Service automation related data view	The preparation of policies	-	BankCon10, BankCon7	HomeCon7, HomeCon9
	The specification of policies	SDCACon3	BankCon8, BankCon9, BankCon11, BankCon13	HomeCon8
	Human participation	-	BankCon11, BankCon14	HomeCon10

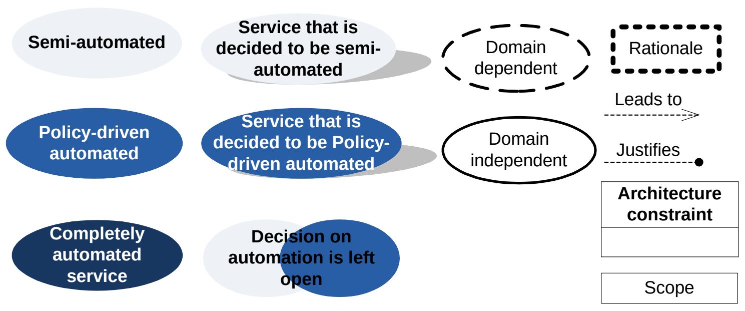

Semi-automated
Service that is decided to be semi-automated
Domain dependent
Rationale
Leads to
Policy-driven automated
Service that is decided to be Policy-driven automated
Domain independent
Justifies
Architecture constraint
Completely automated service
Decision on automation is left open
Scope

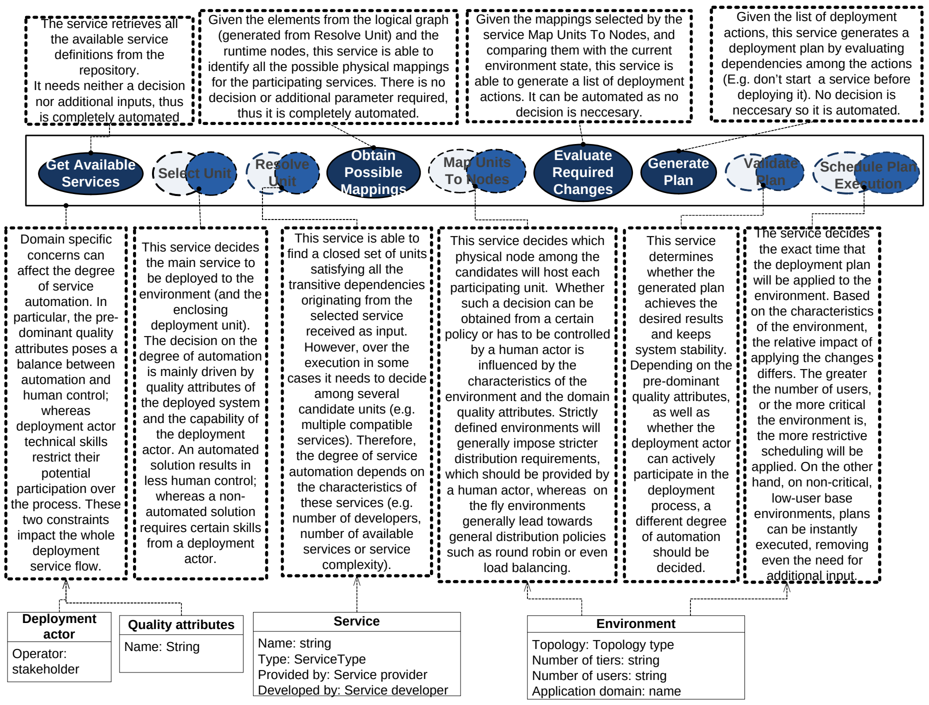
Fig. 2. The graphic notations for the automation decision view

service automation. These services indicate that they can be re-configured to an alternative degree of service automation if necessary. These two sets of notations are meant to address the concerns of *decision on the degree of automation* and reconfigurability in terms of automation.

The two notations in the third column indicate the dependency between a degree of service automation and a specific domain. The notations in the last column are used to illustrate the relation between decisions, architecture constraints, and associated rationale, as well as the scope of services where architecture constraints may have impact on. These two set of notations are meant to address the concerns of *the impact of architecture constraints on the degree of automation*.

4.1 The automation decision view for the SDCA

The automation decision view for the SDCA is presented in Fig. 3. This view aids the SOA architect in taking design *decisions on service automation* by making explicit which architecture constraints may impact the degrees of automation, and which services are affected by each constraint.



The service retrieves all the available service definitions from the repository. It needs neither a decision nor additional inputs, thus is completely automated

Given the elements from the logical graph (generated from Resolve Unit) and the runtime nodes, this service is able to identify all the possible physical mappings for the participating services. There is no decision or additional parameter required, thus it is completely automated.

Given the mappings selected by the service Map Units To Nodes, and comparing them with the current environment state, this service is able to generate a list of deployment actions. It can be automated as no decision is neccesary.

Given the list of deployment actions, this service generates a deployment plan by evaluating dependencies among the actions (E.g. don't start a service before deploying it). No decision is neccesary so it is automated.

Domain specific concerns can affect the degree of service automation. In particular, the pre-dominant quality attributes poses a balance between automation and human control; whereas deployment actor technical skills restrict their potential participation over the process. These two constraints impact the whole deployment service flow.

This service decides the main service to be deployed to the environment (and the enclosing deployment unit). The decision on the degree of automation is mainly driven by quality attributes of the deployed system and the capability of the deployment actor. An automated solution results in less human control; whereas a non-automated solution requires certain skills from a deployment actor.

This service is able to find a closed set of units satisfying all the transitive dependencies originating from the selected service received as input. However, over the execution in some cases it needs to decide among several candidate units (e.g. multiple compatible services). Therefore, the degree of service automation depends on the characteristics of these services (e.g. number of developers, number of available services or service complexity).

This service decides which physical node among the candidates will host each participating unit. Whether such a decision can be obtained from a certain policy or has to be controlled by a human actor is influenced by the characteristics of the environment and the domain quality attributes. Strictly defined environments will generally impose stricter distribution requirements, which should be provided by a human actor, whereas on the fly environments generally lead towards general distribution policies such as round robin or even load balancing.

This service determines whether the generated plan achieves the desired results and keeps system stability. Depending on the pre-dominant quality attributes, as well as whether the deployment actor can actively participate in the deployment process, a different degree of automation should be decided.

The service decides the exact time that the deployment plan will be applied to the environment. Based on the characteristics of the environment, the relative impact of applying the changes differs. The greater the number of users, or the more critical the environment is, the more restrictive scheduling will be applied. On the other hand, on non-critical, low-user base environments, plans can be instantly executed, removing even the need for additional input.

Fig. 3. The automation decision view for the SDCA

This view differentiates the degrees of service automation that are domain-dependent from the ones that are domain-independent. More specifically, the four services that are completely automated are circled with a straight edge line, indicating that they are domain-independent in terms of automation (`SDCACon7`). The other five services, however, are designed being both semi-automated and policy-automated. As such, the SDCA offers its users the flexibility to decide on which degree of service automation to be configured for specific domains, based on domain-specific constraints, such as quality attributes or characteristics of the execution environment (`SDCACon7`).

As an example of domain-independent decision, the deployment service Generate Plan automatically sorts a list of operations, ensuring they are executed in a correct order. The execution of this service only requires the input provided by the previous deployment service Evaluate Required Changes. Hence, the deployment service Generate Plan can be completely automated, independent with any domain specific constraints.

As an example of domain-dependent decision, the service Map Units to Nodes decides the physical distribution of the participating services, among a list of potential mappings provided by the service Obtain Possible Mappings. Depending on the specific domain characteristics, the criteria for making those decisions will be different, as well as the relevance of this decision (ranging from any solution is acceptable to only one distribution is correct). Because of those factors, this service has been designed with flexibility on its degree of automation.

From these examples, we can see that the SOA architect can use this view to explain why some of the services have been designed to be completely automated while others have been designed for both semi-automated services and policy-driven automated services (`SDCACon1`) and the SOA manager is able to use this view to trace, verify and control these decisions (`SDCACon4`).

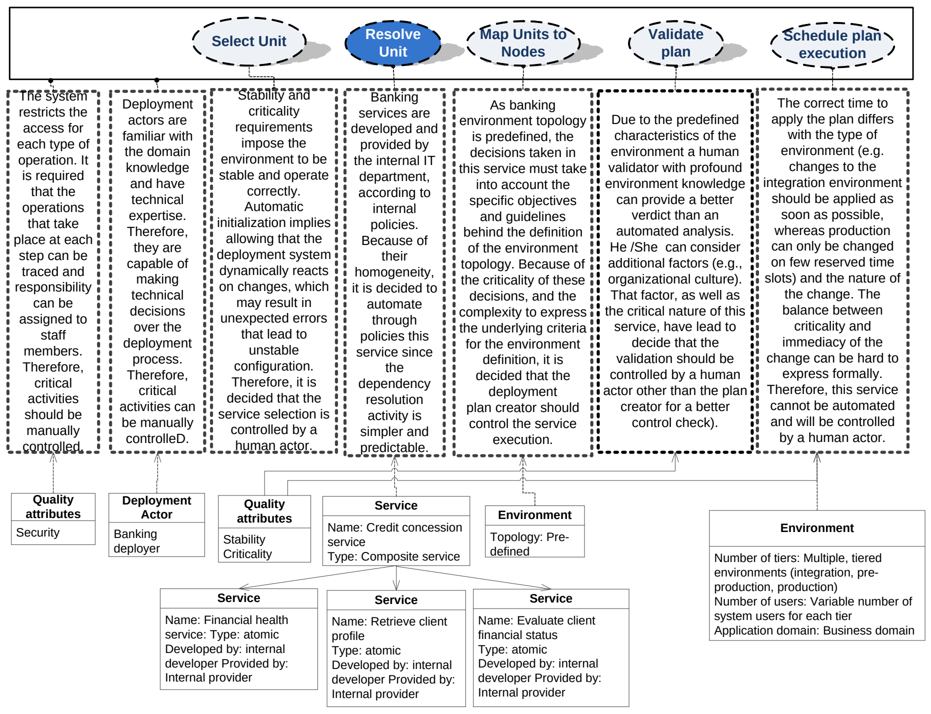
Highlighting the links between the architecture constraints and the decisions, this view facilitates the SOA architect to show the flexibility of adapting the SDCA to specific domain applications. It is obvious from the view that the services whose decisions on service automation are left open, require further re-configuration when architecture constraints become specific (`SDCACon2, SDCACon6`). The users of the SDCA also benefit from this view by being aware of the impact of certain architecture constraints on the degree of service automation.

For instance, the deployment service Select Unit aims at selecting the main service to be deployed to the environment (and the enclosing deployment unit). The decision on the degree of automation is mainly driven by quality attributes of the deployed system and the technical capabilities of the deployment actor. An automated solution results in less human control; whereas a non-automated solution requires certain skills from a deployment actor.

To give another example, the deployment service Map Units To Nodes decides which physical node among the candidates will host each participating unit. Whether such a decision can be obtained from a certain policy or has to be controlled by a human actor is influenced by the characteristics of the environment and the domain quality attributes. Strictly defined environments will generally impose stricter distribution requirements, which need to be provided by a human actor, whereas the environments defined on-the-fly generally lead towards programmatic distribution policies such as round robin or even load balancing.

4.2 The automation decision view for BankFutura

When applying the SDCA to BankFutura, the four completely-automated services whose automation state is domain-independent require no further decisions and hence remain being completely-automated. On the other hand, the five services implemented as both semi-automated and policy-driven automated in the SDCA require further decisions on re-configuration based on the BankFutura specific architecture constraints. The outcome of those decisions is illustrated in the automation decision view for BankFutura, presented in Fig. 4.



Select Unit
Resolve Unit
Map Units to Nodes
Validate plan
Schedule plan execution
Quality attributes
Security
Deployment Actor
Banking deployer
Quality attributes
Stability
Criticality
Service
Name: Credit concession service
Type: Composite service
Environment
Topology: Pre-defined
Environment
Number of tiers: Multiple, tiered environments (integration, pre-production, production)
Number of users: Variable number of system users for each tier
Application domain: Business domain
Service
Name: Financial health service: Type: atomic
Developed by: internal developer Provided by: Internal provider
Service
Name: Retrieve client profile
Type: atomic
Developed by: internal developer Provided by: Internal provider
Service
Name: Evaluate client financial status
Type: atomic
Developed by: internal developer Provided by: Internal provider

The system restricts the access for each type of operation. It is required that the operations that take place at each step can be traced and responsibility can be assigned to staff members. Therefore, critical activities should be manually controlled.

Deployment actors are familiar with the domain knowledge and have technical expertise. Therefore, they are capable of making technical decisions over the deployment process. Therefore, critical activities can be manually controlleD.

Stability and criticality requirements impose the environment to be stable and operate correctly. Automatic initialization implies allowing that the deployment system dynamically reacts on changes, which may result in unexpected errors that lead to unstable configuration. Therefore, it is decided that the service selection is controlled by a human actor.

Banking services are developed and provided by the internal IT department, according to internal policies. Because of their homogeneity, it is decided to automate through policies this service since the dependency resolution activity is simpler and predictable.

As banking environment topology is predefined, the decisions taken in this service must take into account the specific objectives and guidelines behind the definition of the environment topology. Because of the criticality of these decisions, and the complexity to express the underlying criteria for the environment definition, it is decided that the deployment plan creator should control the service execution.

Due to the predefined characteristics of the environment a human validator with profound environment knowledge can provide a better verdict than an automated analysis. He /She can consider additional factors (e.g., organizational culture). That factor, as well as the critical nature of this service, have lead to decide that the validation should be controlled by a human actor other than the plan creator for a better control check).

The correct time to apply the plan differs with the type of environment (e.g. changes to the integration environment should be applied as soon as possible, whereas production can only be changed on few reserved time slots) and the nature of the change. The balance between criticality and immediacy of the change can be hard to express formally. Therefore, this service cannot be automated and will be controlled by a human actor.

Fig. 4. The automation decision view for BankFutura

The view shows how the domain non-functional requirements such as criticality or reliability limit the degrees of service automation that BankFutura actually can operate with, in spite of the advantages of a completely automated service execution flow. As a result, the SOA architect decided to semi-automate most of the services to guarantee a certain degree of control over the deployment process (`BankCon1, BankCon2`). The only service that was decided be policy-driven automated is the deployment service Resolve Unit. An exception was made in that case because BankFutura services are developed and provided by the internal IT department, satisfying the business needs of the organization and are developed according to internal policies. This suggests a simpler and predictable dependency resolution activity and hence it was decided to be automatically driven by policies rather than human actors.

As we can see the requirements of criticality or reliability as well as their impact on the degree of service automation are highlighted in the view (`BankCon2`). Not only the SOA architect can use this view to justify their decisions on the degree of service automation satisfying the requirements of criticality or reliability, but also the deployment manager can use it to trace, verify and control the decisions that the SOA architect made (`BankCon4`).

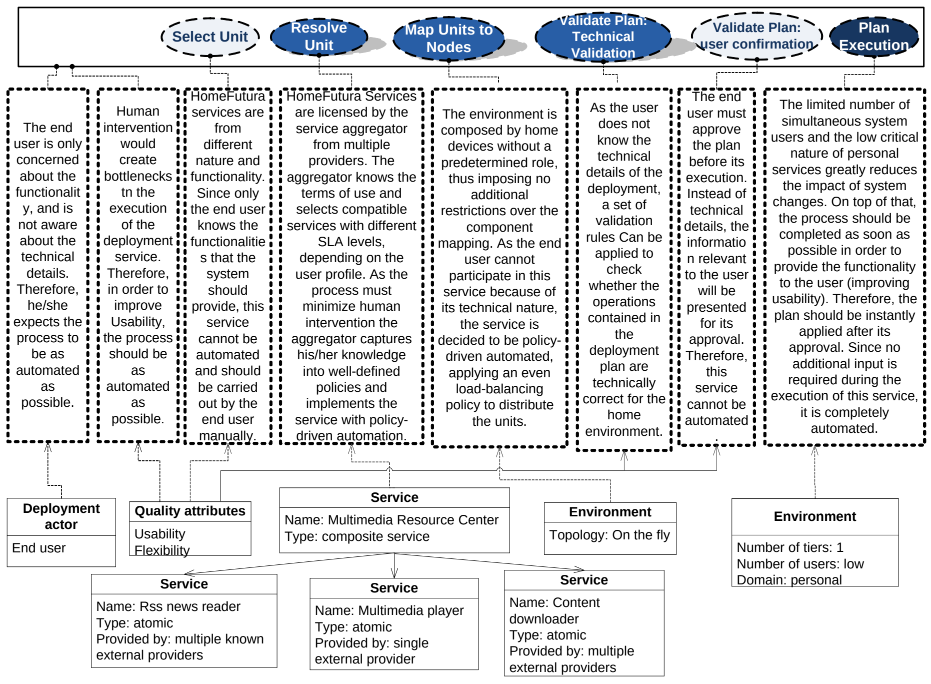
Explicitly documenting the rationale for the decisions on the degrees of service automation also enables the analysis of the possible alternatives on the degree of service automation, allowing to evaluate how the deployment service flow can react to changing requirements or constraints (`BankCon3`). If BankFutura intends to reconfigure the degrees of service automation, it will be useful to know the automation alternatives and the trade-off among them. For instance, the services presented in Fig. 4 are marked with a shadow if they can be either semi-automated or policy-driven. Although the services have been decided to implemented with either of the degrees of service automation, they could be reconfigured to another degree if the organization deemed it necessary (e.g. after a time of operation the organization increased its trust in the automation capabilities of the BankFutura, and opted to increase the degree of automation for a more efficient operation).

4.3 The automation decision view for HomeFutura

Specialized from the degree of automation view for SDCA (presented in Fig. 5), this view presents the implemented degree of automation for HomeFutura services.

Similar to BankFutura, the degree of automation view for HomeFutura (presented in Fig. 5) does not display the four completely automated services that are domain independent and need no further decisions. This view emphasizes the decisions on the degree of automation for the other five services that are domain-dependent, as well as the domain specific constraints that lead to these decisions.

Whereas in BankFutura environment stability is the dominant constraint, HomeFutura aims at providing end users the flexibility to experience new services available on the network. Consequently, HomeFutura opted for a deployment solution with higher degree of automation, not only because end users are not capable of providing technical input to the deployment process but also because



The end user is only concerned about the functionality, and is not aware about the technical details. Therefore, he/she expects the process to be as automated as possible.

Human intervention would create bottlenecks tn the execution of the deployment service. Therefore, in order to improve Usability, the process should be as automated as possible.

HomeFutura services are from different nature and functionality. Since only the end user knows the functionalities that the system should provide, this service cannot be automated and should be carried out by the end user manually.

HomeFutura Services are licensed by the service aggregator from multiple providers. The aggregator knows the terms of use and selects compatible services with different SLA levels, depending on the user profile. As the process must minimize human intervention the aggregator captures his/her knowledge into well-defined policies and implements the service with policy-driven automation.

The environment is composed by home devices without a predetermined role, thus imposing no additional restrictions over the component mapping. As the end user cannot participate in this service because of its technical nature, the service is decided to be policy-driven automated, applying an even load-balancing policy to distribute the units.

As the user does not know the technical details of the deployment, a set of validation rules Can be applied to check whether the operations contained in the deployment plan are technically correct for the home environment.

The end user must approve the plan before its execution. Instead of technical details, the information relevant to the user will be presented for its approval. Therefore, this service cannot be automated

The limited number of simultaneous system users and the low critical nature of personal services greatly reduces the impact of system changes. On top of that, the process should be completed as soon as possible in order to provide the functionality to the user (improving usability). Therefore, the plan should be instantly applied after its approval. Since no additional input is required during the execution of this service, it is completely automated.

Fig. 5. The automation decision view for HomeFutura

agility in the execution is required. The view presents this rationale and shows its link with the architecture constraints(`HomeCon2` influencing the decisions.

As a result, only two services that require input from end users are semi-automated. The service Select Unit requires the end users to select the services that they would like to experience; and the other service Validate Plan requires the end users to approve the execution of the deployment plan (e.g., the cost of news service, the changes to multimedia player device). The rest of the services in the deployment process are all automated and do not require human intervention . Similar with the decision view for BankFutura, this view enables the SOA architect to justify the decisions and supports the deployment manager in governing the decisions (`HomeCon1, HomeCon5`).

It is worth to note that three services are marked with a shadow, which indicates that a choice has been made between semi-automation and policy-driven automation for these services. Although policy-driven automation has been chosen for the current deployment solution for HomeFutura, the shadow reminds the architect that this service could be re-configured to be semi-automated if needed (`HomeCon3`). For instance, after a time of operation it turns out the service Map Units To Nodes does not provide the most optimal mapping of the selected services to devices (due to e.g. too complicated dependency graph that Map Units To Nodes cannot interpret correctly or incomplete policy that is not able to provide sufficient information for the mapping), the architect could decide to let a technical expert decides the mapping and hence make Map Units To Nodes to be semi-automated. However, involving another deployment actor (other than the end user) in the process would cause that the user cannot instantly start to experience the new service, having to wait for the required input from the technical expert. This way, usability and agility of HomeFutura would be challenged.

5 The degree of service automation view

Whereas the automation decision view emphasises the domain-specific constraints that lead to the decisions on the degree of service automation, the degree view focuses on the decided *degrees of service automation* for the service execution flow. As a result, only the degree of service automation that the service flow is expected to achieve is relevant in this view, while the details of how the decisions are made are irrelevant and should not be presented in this view.

As denoted by the graphic notations presented in Fig. 6, the degrees of automation are graphically rendered by the darkness of the color assigned to each service: the darker is the color, the higher is the degree of automation. In addition, human actors are associated to semi-automated services with the purpose of highlighting who are expected to provide input to which services. The sequence between services indicates the order in which the deployment services are invoked. With this additional information, the period during which human intervention is (and is not) required becomes explicit. By illustrating that external inputs are expected to be provided by *whom* and *when*, this view also addresses the *accountability*.

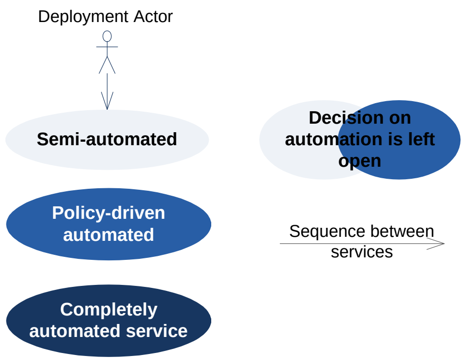

Deployment Actor
Semi-automated
Decision on automation is left open
Policy-driven automated
Sequence between services
Completely automated service

Fig. 6. The graphic notation for constructing the degree of automation view

5.1 The degree of service automation view for the SDCA

Applying the graphic notations presented in Fig. 6, we constructed the degree of service automation view for the SDCA shown in Fig. 7.

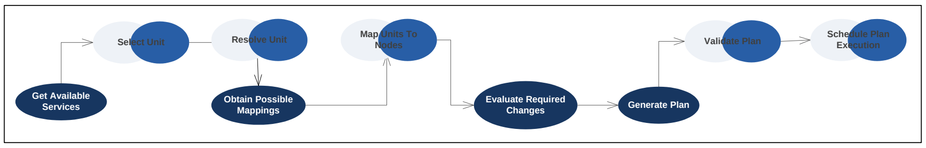


Fig. 7. The degree of automation view for the SDCA

Using this view, the SOA manager can gain an overview of the degree of service automation supported by the SDCA, as a result of the decisions illustrated in Fig. 3. More specifically, two different degrees of automation are designed for the SDCA (`SDCACon5`). Deployment services that do not require additional information are completely automated, while those needing external information are designed to retrieve the external information either from human actors or from policies.

5.2 The degree of automation view for BankFutura

Analogous to the SDCA, we also used the described notation to construct the degree of service automation view for BankFutura, as it can be seen in Fig. 8.

The deployment manager can see from this view the three degrees of service automation designed for the BankFutura (BankCon5). More specifically, four services are completely automated, one is policy-driven automated and four are semi-automated. In other words, this view shows that nearly half of the services require human intervention, meaning that the automation degree of the deployment process for BankFutura is relatively low.

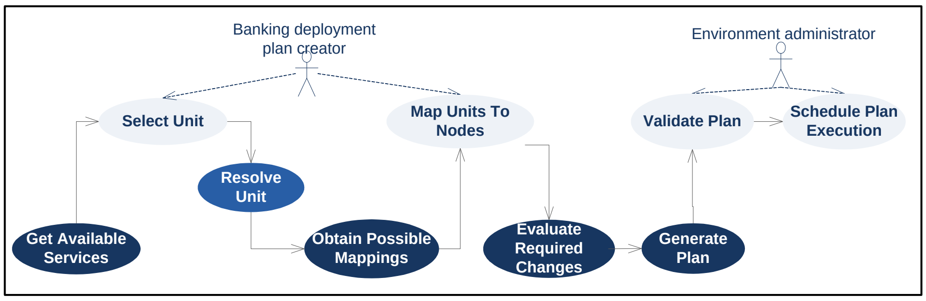


Fig. 8. The degree of automation view for BankFutura

In addition, as the semi-automated services are associated with a banking deployment plan creator and an environment administrator, the manager may use this view to trace the responsibility of these two human actors who are expected to provide input during the services execution (BankCon6).

Similarly, this view points out directly for the banking deployment plan creator and environment administrator which services are expecting their input. As shown in Fig. 8, the role of the banking deployment plan creator is associated to the deployment services Select Unit and Map Units To Nodes, indicating that as soon as the deployment service flow is initiated and the banking deployment creator should be prepared to first make a selection on the available units and later on to establish mapping between selected units and physical nodes (BankCon14). On the other hand, the environment administrator is only involved in the validation and execution of the deployment plan (BankCon11).

5.3 The degree of automation view for HomeFutura

Similar with the degree of automation view created for BankFutura, Fig. 9 shows the designed degree of automation for each service in the deployment process of HomeFutura.

The view visually highlights the fact that a higher degree of automation has been designed for HomeFutura as compared to BankFutura (HomeCon4, HomeCon6). As shown in Fig. 9, most of the services are illustrated with dark color (indicating that the services can execute without any human intervention); while two services are in light color (indicating that human intervention is needed).

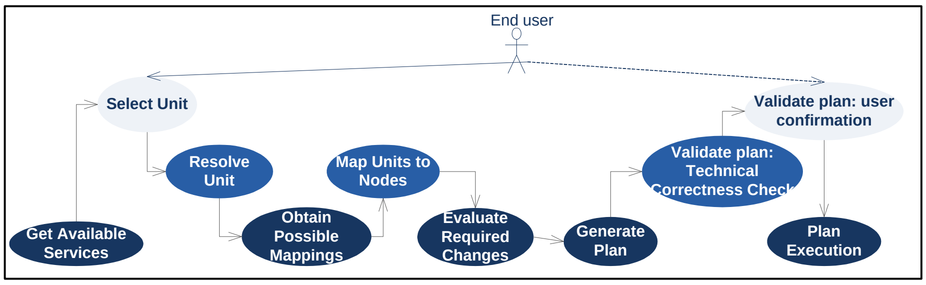


Fig. 9. The degree of automation view for the HomeFutura

The degree of service automation view for HomeFutura can also be used to explain to the end users how they are expected to participate in the deployment process. End users can see hwo their participation consists of selecting the home services that they would like to experience using service Select Units and eventually to agree on the corresponding costs of consuming these services by using service Validate Plan. This way, it can be seen how the end users are presented only the relevant information for them, while the low level, technical details are hidden(`HomeCon10`).

6 The automation-related data view

While the *degree of service automation view* highlights the degree of service automation designed for the deployment service flow, the *automation-related data view* details the design from the data perspective. This way, the questions related to the generation, management and provision of additional input (from either human actors or policies) can be answered by the automation-related data view.

The graphic notations that we created to construct the automation-related data view is presented in Fig. 10. Besides the notations for the three degrees of service automation, we distinguish the guidelines/rules from formalized policies. While both guidelines/rules and formalized policies are relevant to service automation, the former are used by the deployment actors to drive the decision and the latter are directly accessed by deployment services to achieve policy-driven automation.

The most right hand side of Fig. 10 shows the graphic notation denoting the relationships between elements in the automation-related data view. More specifically, a deployment actor *is responsible for* providing an input; such input *assists the execution* of semi-automated services; guidelines / rules guide the provision of such an input; formalized policies directly *assists the execution* of policy-driven automated services; and *sequence between services* is also denoted. Given these details on the relationships between policies, deployment actors, and deployment services, the deployment actors can tell which services are expecting

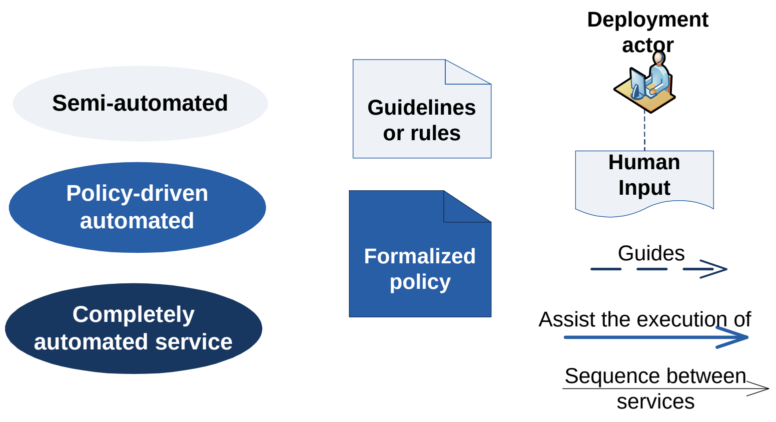


Fig. 10. The graphic notation for constructing the automation-related data view

what information from them. Moreover, it explicitly points out which organizational guidelines or rules should this information comply with. In this way, the deployment actors can be prepared to transform this organizational knowledge to their input to services, hence facilitating *human participation*.

In addition to the graphic notation, we also constructed a table template (shown in Tab. 5) for listing the policies that are relevant to service automation in the deployment service flow. This table aids *the specification of policies* in presenting all the information relevant to the policies in a structured manner. As such, this table also aids the *preparation of the policies*.

Table 5. The template for automation-related policy table

Policy ID	Policy name	Policy Description	Associated service	Controlled by	Type of format

6.1 The automation-related data view for the SDCA

As the SDCA is a reference deployment management system, it does not define concrete deployment actors or policies, as they will be specialized in specific applications. However, in order to guide the application of the SDCA, the SOA architect is concerned about identifying the required policies for providing the additional input to the deployment service flow. For this reason, we constructed the automation-related policy table, using the template presented in Tab. 5.

The *automation-related policy table* (presented in Table 6) provides detailed information about all the policies, including the ones for guiding human actors and the ones for policy-driven automated services. Using this table, the SOA architects in specific domains can gain an overview on what type of policies might be relevant and should be prepared, as well as which format should they be expressed when applied to the SDCA (`SDCACon3`).

Table 6. The automation-related policy model for the SDCA

Policy ID	Policy name	Policy Description	Associated service	Controlled by	Type of format
P01	System requirements	Describes the functionality (services) that the target environment must provide	Select Unit	-	Formal/Text
P02	Unit selection policy	Provides criteria for selecting among multiple candidate units that satisfy the same dependency	Resolve Unit	-	Formal/Text
P03	Unit distribution policy	Provides criteria for selecting the physical place of the environment where each deployment unit will be installed	Map Units To Nodes	-	Formal/Text
P04	Plan Validation Rules	Defines a set of checks that can identify fatal errors in the plan definition or potential risks expressed as warnings	Validate Plan	-	Formal/Text
P05	Environment update policy	Controls at what periods of time plans can be applied to the environment	Schedule Plan Execution	-	Formal/Text

6.2 The automation-related data view for BankFutura

Applying the graphic notation presented in Fig. 10, we constructed the automation-related data view for BankFutura (shown in Fig. 11). This information is complemented with the BankFutura policy table, presented in 7)

From this view the two human actors, the *banking deployment plan creator* and the *environment administrator*, can see what types of information they are expected to provide to which services during the service execution flow (`BankCon11, BankCon14`). In addition, they can see which organization policies (or guidancerules) can be referred in order to provide the required information (`BankCon9, BankCon11, BankCon12, BankCon13`).

We will use an example scenario to illustrate how `BankCon10` is supported by the data view, guiding the participation of the *deployment plan creator* in the service *Map Units to Nodes*. In an specific environment, the environment design document mentioned in the data view informs that only one server from the environment infrastructure is configured in the network firewall to be remotely accessible from the outside; the remaining elements being protected from outer clients. In this case, by analyzing that information, the human will decide to assign units containing final services (which must be remotely accessible) to the visible server, whereas the remaining elements will be distributed over the other elements, regardless of whether those servers might also be technically capable of hosting the same types of services.

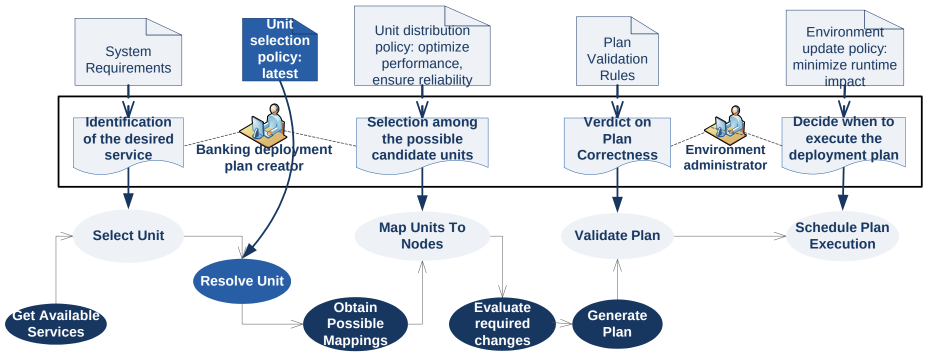

System Requirements
Unit selection policy: latest
Unit distribution policy: optimize performance, ensure reliability
Plan Validation Rules
Environment update policy: minimize runtime impact
Identification of the desired service
Banking deployment plan creator
Selection among the possible candidate units
Verdict on Plan Correctness
Environment administrator
Decide when to execute the deployment plan
Select Unit
Resolve Unit
Map Units To Nodes
Validate Plan
Schedule Plan Execution
Get Available Services
Obtain Possible Mappings
Evaluate required changes
Generate Plan

Fig. 11. The automation-related data flow view for BankFutura

Fig. 11 shows that the policy-driven automated service is explicitly linked to the corresponding policy. For instance, service Resolve Unit is linked to *Unit selection policy* indicating that the criteria for selecting among multiple candidate units defined by the policy directly influences the output of Resolve Unit. Explicitly visualizing the dependency between the services and their corresponding policies, the automation-related data flow model aids the deployment manager in obtaining an overview on when certain policies are required and which services require them during the deployment service flow (`BankCon7`).

Complementary to the data view, the automation-related policy table further details each policy presented in Fig. 11 by noting its description, the role that is in charge of it and the type of format for documenting the policy. This table aids the preparation of policies as it can be used as a check list for the deployment manager to assign tasks to some specific personnel, making sure that the policies are in place and are expressed in the right format before the execution of the deployment process (`BankCon7`).

6.3 The automation-related data view for HomeFutura

Applying the graphic notation shown in Fig. 10, the automation-related data view for HomeFutura is presented in Fig. 12. From this view, the end users can see that they are only required to initiate the deployment process when they want to experience new services and confirm the operation when the selected services are ready to be deployed. More importantly, the end users will be confident in providing this information as the data view shows that the former is based on their own functional requirements and the latter on their own non-functional requirements. With limited involvement in the deployment process, the end user has full control over the selection of new home services and the acceptance of any associated cost (`HomeCon10`).

As the deployment process for HomeFutura has much higher degree of automation than our previous case, it is more critical that the policies are in place before the deployment process starts as compared to the case of BankFutura.

Table 7. The automation-related policy model for BankFutura

Policy ID	Policy name	Policy Description	Associated service	Controlled by	Type of format
PB01	System requirements	Describes the business processes that must be supported by the environment	Select Unit	Managers	Textual Document
PB02	Unit selection policy: Latest version	Selects the unit with the most recent version among the potential candidates	Resolve Unit	SOA Architects	Formal/SQL Sorting Query
PB03	Unit distribution policy	Defines the rationale behind the environment definition and the quality levels to be sustained by the deployed services	Map Units To Nodes	SOA Architect	Textual Environment design / SLA Document
PB04	Plan Validation Rules	Checks whether the plan is coherent with the current state of the environment	Validate Plan	Environment Administrator	Formal/RETE-based rules + Guidelines Textual Document
PB05	Environment update policy: Minimize runtime impact	Controls the reserved time slows for applying changes, and avoids overlap in the execution of concurrent changes	Schedule Plan Execution	Environment Administrator	Environment Operation Guidelines Textual Document + Environment Calendar

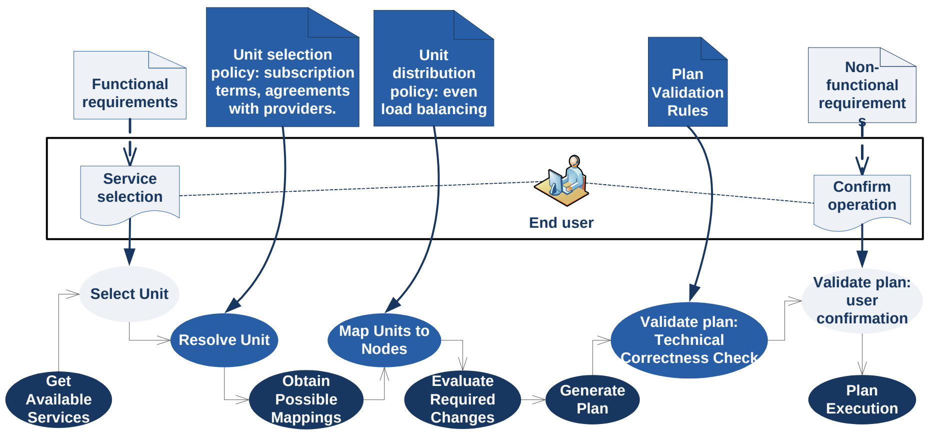

Functional requirements
Unit selection policy: subscription terms, agreements with providers.
Unit distribution policy: even load balancing
Plan Validation Rules
Non-functional requirements
Service selection
End user
Confirm operation
Select Unit
Resolve Unit
Map Units to Nodes
Validate plan: Technical Correctness Check
Validate plan: user confirmation
Get Available Services
Obtain Possible Mappings
Evaluate Required Changes
Generate Plan
Plan Execution

Fig. 12. The automation-related data flow view for HomeFutura

In BankFutura, there is only one policy-driven service requiring direct access to its associated policy. The other semi-automated services require the associated policies to be well documented enough so that the deployment actors are able to provide the required input. In HomeFutura, as shown in Fig. 12, there are three policy-driven services, which means that the three associated policies should all be accessible before the deployment process starts (HomeCon7).

Complementary to the data view, the automation-related policy table for HomeFutura presented in Tab. 8 provides detailed information about the policies illustrated in the data view. From this table, the service aggregator can see that he/she is the main role who is in charge of the formalized policies (`HomeCon9`).

Services available to HomeFutura come from different service providers, after being licensed by the service aggregator. The aggregator knows about their use licenses and selects compatible services with different SLA levels, depending on the user profile. With the aggregator defining policies for solving dependency conflicts based on the user profile, the technical knowledge of the aggregator is transferred to formalized documents that the three deployment services can directly access to. This not only enables these services to be policy-driven automated, but also ensures the alignment between them and technical requirements that are specific to HomeFutura (`HomeCon8`).

Table 8. The automation-related policy model for HomeFutura

Policy ID	Policy name	Policy Description	Associated service	Controlled by	Type of format
PH01	Functional Requirements	Functionality that must be provided from the HomeFutura platform to the end user	Select Unit	End User	Undocumented knowledge
PH02	Unit selection policy	Selects among the candidate units based on the agreements with service providers and the user subscription terms	Resolve Unit	Service aggregator	Subscription Terms Document, Agreements with providers document
PH03	Unit distribution policy: Even load balancing	Distributes the affected units over the home environment attempting to balance the load on the computing nodes	Map Units To Nodes	Service aggregator	Formal: Linear Programming coding
PH04	Plan Validation Rules	Checks whether the plan is coherent with the current state of the environment and the subscription terms of the end user	Validate Plan: Technical Correctness Check	Service aggregator	Subscription Terms + Formal/ RETE-based rules
PH05	Non-functional requirements	Checks whether the consumption of the selected service complies the non-functional requirements of the end user	Validate Plan: User Confirmation	End user	Undocumented knowledge
PH06	Environment update policy: Instant execution	Instantly applies the plan to the home environment	Schedule Plan Execution	Service aggregator	Formal: code

7 The power of visualization

Visualization is a common technique to convey abstract information in intuitive ways. Representing information in terms of (a set of) graphics often more easily draws readers' attention and improves understandability, as compared to pieces of text. That is why visualization has been considered as one of the most effective ways for communication. In the same vein, effectively applying the technique of visualization in architecture description improves the communication between stakeholders; in our experience the range of stakeholders involved in services engineering is broader than in the development of traditional applications, rendering the usage of visualization techniques as a key to get effective communication. On the other hand, the usage of diagrams (such as UML) well-known in the field of software architecture, while useful for technical stakeholders, results to be difficult to understand and reason with for non-technical ones (such as users in HomeFutura).

When constructing the service automation views, we consciously design the graphic notation to make the views intuitively understandable and to hold readers attention steady. Some of these graphic notation have been commonly used in architecture description, like using a symbol of person to stand for a human actor, or using a symbol of document to stand for a policy.

In addition to these commonly used notation, we created a set of color schema representing the different degrees of service automation. The motivation behind this schema is that from the human perception point of view, objects with dark color often make one feel heavy in weight (and easy to sink), whereas objects with light color make on feel light (as easy to float). As shown in the views, the degree of automation is graphically rendered by the darkness of the color assigned to each service: the darker the color, the higher s the degree of automation. This representation resembles an iceberg immersed in the sea: only the top (white is the lightest color) is visible (i.e. the user is aware of the service and manually participates in its execution), while the deeper the iceberg is sunk in water, the lesser visible it becomes (i.e. accessible by users, or in other words, increasingly automated).

In the same vein, the graphic notation for the policies inherit the same color schema. As shown in Fig. 10, the policies for providing guidance for deployment actors are in light color; whereas the policies for assisting the execution of policy-driven automated services are in dark color. As such, from the color of the policies shown in Fig. 11, and 12, the reader can have the perception of the correspondence between the degree of service automation and policies. In addition, in both the degree of service automation view (shown in Fig. 7, 8, and 9) and automation-related data view (shown in Fig. 11, and 12, the completely-automated services with dark color "sink" at the bottom, implying that they are loosely coupled with human actors. Whereas the semi-automated services with light color "float" at the top, implying that they are tightly coupled with human actors. As such, these visualization techniques enable the views become self-explaining.

8 Observation

By studying the SDCA, and in particular its two industrial cases, we have identified a set of concerns that are particularly relevant to service automation. By constructing views to illustrate the service automation aspect, we gained insight into the way in which service automation has been designed under different contexts. In addition, during the course of this work, we made several observations.

First, we noticed that the degree of service automation is also relevant to the level of service granularity, which was not foreseen in this study. In the design of SBAs, the appropriate level of granularity of services is often regarded of great importance and challenging. The alignment between business and IT is often the (only) main driver for service identification due to the benefits it might bring [11], such as the ease of comprehension of the design, the increase of potential reuse, just to name a few. Since service granularity in nature does not share common interests with service automation, it was not identified as one of the concerns to be addressed by the views. However, in this work we noticed that the service granularity, to certain extent, is also influenced by service automation, which again makes the relevance of service automation to SOA more evident.

In the case of HomeFutura, we noticed that the deployment service Validate Plan was decomposed to two services, one is Validate plan: Technical Correctness Check and another is Validate plan: user confirmation. The main driver for this decomposition is that Validate Plan consists of two types of validation that can be implemented with two different degrees of automation. The technical validation aims to check whether the operations contained in the deployment plan are technically correct for the home environment. This can be done automatically, provided that the policy *Plan Validation Rules* is available. Whereas the user validation aims to get approval from the end users if they agree with the non-functional attributes associated to the select home service. Since the end users are the only ones that have the knowledge on their own preferences and these preferences vary from person to person, it is not feasible to embrace this knowledge into a policy and it has to be controlled directly by the end users. As a result, the deployment service Validate plan: Technical Correctness Check validates the deployment plan from a technical perspective and is designed to be policy-driven automated; whereas the deployment service Validate plan: user confirmation validates the deployment plan from a user perspective and is designed to be semi-automated.

In the case of BankFutura, the deployment service Validate Plan was not decomposed although it also consists of the technical validation and user (system) validation. Similar with HomeFutura, *Plan Validation Rules* can be formalized as a policy that Validate Plan can directly access. The difference lies in the fact that the non-functional requirement of the system is known by the environment administrator and hence can also be formalized as a policy. In this way, Validate Plan can be designed as policy-driven automated, accessing *Plan Validation Rules* that consists of both the rules for technical validation and the non-functional requirements.

From these two examples, we can see that the identification of the deployment services or the level of service granularity is not only driven by the business functionalities that they represent, but also influenced by the degree of service automation. Despite the benefits that the business-IT alignment may achieve, an architect sometime would decompose a coarse-grained service into multiple fine-grained services due to different service automation requirements. The result of decomposition might lead to a SBA with higher maintainability and adaptability in terms of service automation but tightly-coupled services and decreased reusabiliy. As such, in the design of SBAs, an SOA architect has to make a tradeoff between the alignment with business functionalities and the level of service automation.

The second observation we made is on the applicability of the views to architecture descriptions of SBA in general. As explained by Shull et. al [12], the sources of variability may influence the result, such as the types of projects for which a technique is effective. For this reason, we did an analysis on the variability of the domain and the type of architecture that we studied.

More specifically, the design of the SDCA aims at providing a reference architecture for service configuration and management while the same time focuses on its applicability in industrial domains. We also studied the application of the SDCA in an enterprise domain and a personal domain with completely different characteristics. The difference between these domains contributes to the variability of this study. Although the concerns elicited from the SDCA and its two case studies are somehow different and represent domain-specific interests, addressing these concerns in architecture description demands for similar types of information. When illustrating all these types of information in terms of the same set of graphic pictures and tables (or views), we are able to show that all the concerns identified from each case have been addressed in the corresponding architecture design. As a result, we are confirmed that the views can be applied to three different domains.

However, the concerns that we identified are all related to the SDCA, both its own design and its applications in different industrial domains. The lack of variability in terms of the type of architecture that we studied might threat the validity of the views in illustrating the service automation aspect of SOA in general. For this reason, we plan to replicate the study by analyzing the service automation aspect of various types of SOA in our future work.

9 Conclusion

In this chapter we have studied the different degrees of automation suitable for services configuration and deployment on different domains (business and home). While the initial goal was just the development of a system able to perform these functions -a Services Deployment and Configuration Architecture or SDCA- we discovered that the architectural concerns were affected by the specific domain of application it was to be used for; in fact there are several quality attributes that must be covered, but the balance between trust and reliability for example, is specific to the domain.

The key contribution of this paper is the identification of three views that structure and ease elicitation and documentation of stakeholders' concerns. The application of these three views onto the bank and the home domain case studies clearly reflects the differences on the degree of automation for a similar set of basic functions (provided by the services); with a lower degree of automation at the bank domain when compared to the home domain. The notation we have used for the description of views and decisions is simplified with respect to available notations. This allows for a better representation of the concepts involved in the architectural decision making by stakeholders, while remaining intuitive even for non-technical ones. The expression of usually implicit architectural knowledge allowed us getting a hint on the relationship between the degree of automation and the granularity of services. Also, the usage of the same description technique across domains revealed commonalities between them.

The results obtained seem promising, but in order to better capture the wide variability of service automation we plan as future work to to apply the same process to additional Service-Based Applications, as well as applying the approach to several more unconnected domains. This way, it would also be interesting to validate whether different SBAs belonging to the same domain share specific constraints, which affect their decisions on the degree of automation the same way.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley (2003)
2. Emery, D., Hilliard, R.: Updating ieee 1471: architecture frameworks and other topics. In: WICSA '08: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), IEEE Computer Society (2008) 303–306
3. Heinz-Gerd Hegering, Sebastian Abeck, B.N.: Integrated management of networked systems: concepts, architectures, and their operational application. Morgan Kaufmann Publishers Inc (1998)
4. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer Magazine, IEEE **36**(1) (January 2003) 40–49
5. Ruiz., J.L., Dueñas, J.C., Cuadrado, F.: Model-based context-aware deployment of distributed systems. Communications Magazine, IEEE **47**(6) (June 2009) 164–171
6. Gu, Q., Lago, P.: On service-oriented architectural concerns and viewpoints. In: 8th Working IEEE/IFIP Conference on Software Architecture (WICSA), Cambridge, UK, IEEE (2009)

7. ANSI/IEEE: Standard glossary of software engineering terminology, std-729-1991. ANSI/IEEE (1991)
8. Kruchten, P., Lago, P., van Vliet, H.: Building up and reasoning about architectural knowledge. In: 2nd International Conference on the Quality of Software Architectures (QoSA). (2006)
9. Ali Babar, M., Dingsoyr, T., Lago, P., van Vliet, H., eds.: Software Architecture Knowledge Management: Theory and Practice. Springer (jul 2009)
10. Andrikopoulos, V., Bertoli, P., Bindelli, S., Nitto, E.D., Gehlert, A., Germanovich, L., Kazhamiakin, R., Kounkou, A., Pernici, B., Plebani, P., Weyer, T.: State of the art report on software engineering design knowledge and survey of HCI and contextual knowledge (2008)
11. Heuvel, W.J.v.d., Yang, J., Papazoglou, M.P.: Service representation, discovery, and composition for e-marketplaces. In: Proceedings of the 9th International Conference on Cooperative Information Systems (CoopIS 2001). Volume Lecture Notes In Computer Science; Vol. 2172., Springer-Verlag London, UK (2001) 270284
12. Shull, F., Carver, J., Vegas, S., Juristo, N.: The role of replications in empirical software engineering. Empirical Software Engineering **13**(2) (2008) 211218