Biuletyn WAT
Vol. LXII, Nr 1, 2013

Systems integration with integrating bus in SOA architecture using patterns

DAWID ADACH[1], WŁODZIMIERZ DĄBROWSKI[1], ANDRZEJ STASIAK

Wojskowa Akademia Techniczna, Wydział Cybernetyki,
00-908 Warszawa, ul. S. Kaliskiego 2, astasiak@wat.edu.pl
[1]Politechnika Warszawska, Instytut Sterowania i Elektroniki Przemysłowej,
00-662 Warszawa, ul. Koszykowa 75, dawid@adach.com, w.dabrowski@ee.pw.edu.pl

Abstract. This article describes a method for determining patterns of adapters structures for systems using ESB bus. In the proposed method, we wanted the design of the master adapter structure to be determined by the appropriate division dedicated adapters. Created adapters are strongly connected internally, because the process of canonization allow you to communicate only with the processes in the same domain (functional area). In addition, we assumed that the set standards should lead to the elimination of critical points and bottlenecks of the built system, and increase efficiency in communication and effectiveness of resource management. The method was tested in the Tibco ESB environment, while working on a solution for the telecommunications operator.
Keywords: System integration (SI), EAI, ESB, patterns for ESB, canonical ESB adapters, SOA, UML

1. Introduction

When in the late 70's the IT market started growing rapidly and foundations of Service Oriented Architecture (SOA) (it had not been named so yet) were created, the new problem appeared — the problem of Systems Integration. The following article describes practical approach to the subject of systems integration which is supported by the authors' experience obtained from implementation of Enterprise Service Bus (ESB) in leading Polish telco operator. The article contains review of proposed patterns recommended while building integrating bus, their verification on a real platform, and proposals of their enhancement held by examples which explain the need of standard solutions' expansion. The aim of the following article

is to show common problems which appear during the systems integration process. The article contains an analysis of popular integration problems and different approaches of solving them. It describes problems related to a process of system integration like different communication patterns or transaction activities; basics assumptions, and recommended approach of Service Oriented.

2. Integration

Integration is a very wide term which contains many issues. To start with simple integration of two systems using build-in tools to exchange messages, through complex integration of data and functionality to finish with total integration of almost all systems (and subsystems), databases, joint order management etc.

We can split the integration into four main areas [1]:

— Application integration — all activities connected with implementation of specialized software (the trade software);
— Data integration — activities which purpose is to eliminate duplicates from database and also elements which are redundant;
— Web integration — activities related to installation of local and extensive networks, structural wiring, fibers etc.;
— Systems integration — activities in the field of installation operating systems, databases, office and communication software, structural wiring, active network devices, mass device storage, internet accesses, access control, supply etc.

As definitions above show, integration in IT is very wide and complex term. Systems integration as the largest one and least defined should be treated as a separate type that operates together with other integration types. Therefore, there is no one correct solution (or instruction) — on "how to integrate systems". Each case is individual and all of them should be treated respectively.

We can distinguish three main types of systems integration due to the method of integration. The most popular and the worst from technical point of view — Star Integration, integration of group of systems with similar functionalities — Vertical Integration and actually the best approach but also the most labour-intensive — Enterprise Service Bus (Horizontal Integration).

2.1. Star integration

Star Integration, also known as Spaghetti Integration [2] because of amount of connections, is the most popular type of integration (Fig. 1). Popularity of this solution is a consequence of lack of long-term planning or sudden expansion of the company.

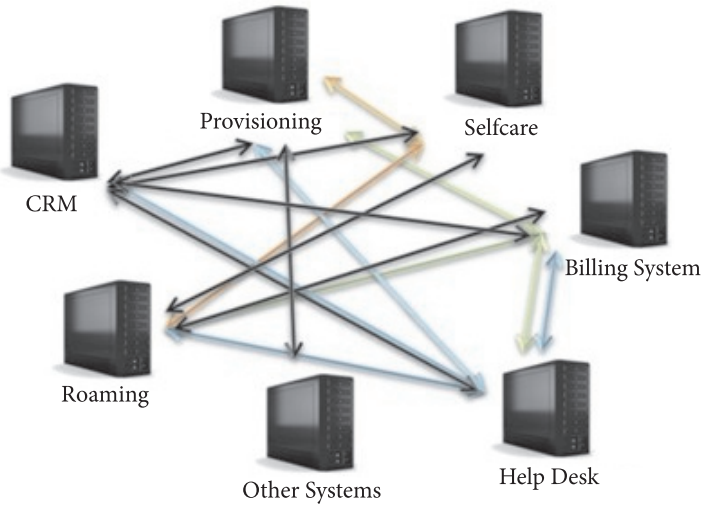

Fig. 1. Visualization of Star Integration

Star Integration is recommended when only a few systems need to be connected or for the companies where implementing SOA is not remunerative. It is also used when majority of systems is supplied by one provider (e.g. SAP) and systems contain built-in tools to integrate with other systems. In other case, each system has to adapt to others by creating dedicated adapters. It means that a system which primarily used to use one protocol (e.g. HTTP) and language (e.g. XML) has to create a new adapter which will support communication patterns used by all other systems to communicate with them.

Management and maintenance of such architecture is also inconvenient. Lack of global preview, distributed and different types of logs and the last but not the least — difficult and expensive extensibility (adding a new system demands creation of new adapters in many systems) causes that Star Integration is not recommended for medium and big companies.

2.2. Vertical integration

Vertical Integration presents different approach to the case (Fig. 2). It is a process of gathering subsystems responsible for similar areas in bigger entities also often called silos. Communication among systems in one area is hidden behind other ones. Communication between areas takes place in dedicated adapters controlled by the process control system responsible for specified area.

The advantage of this approach is that the integration is performed quickly and involves only the necessary vendors therefore this method is cheaper in short term. On the other hand, cost of ownership can be substantially higher

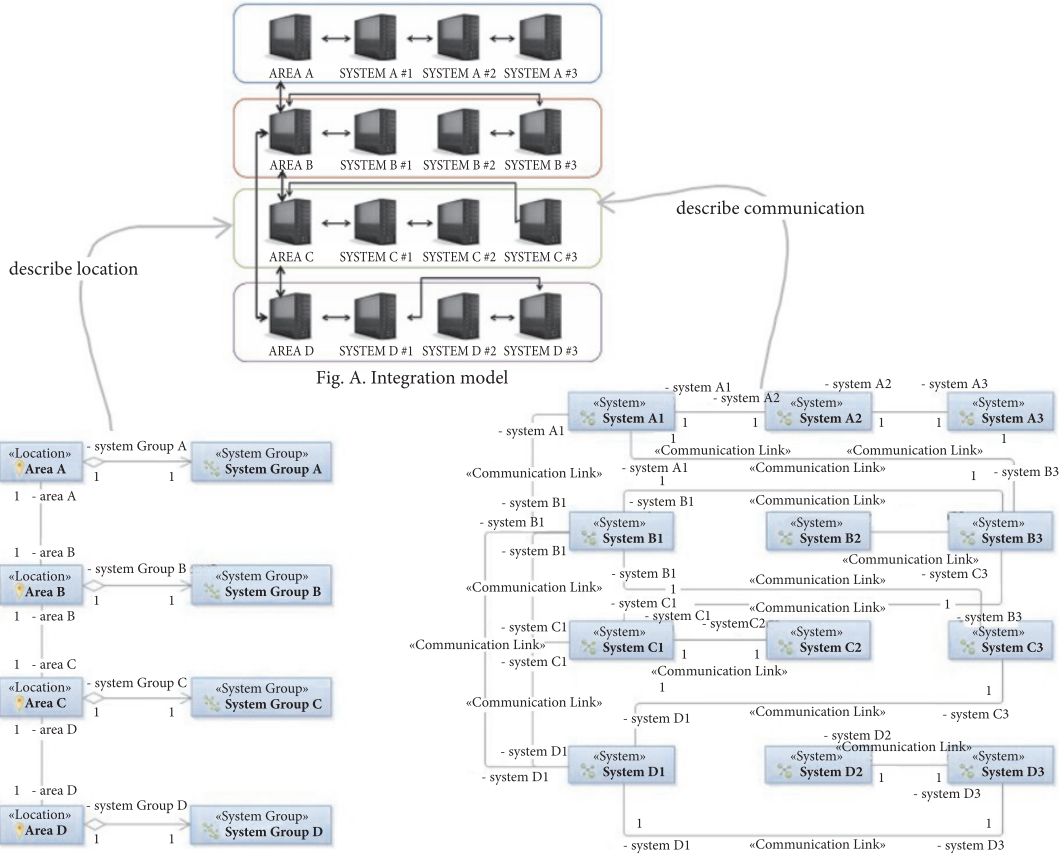

Fig. A. Integration model

Fig. B. UPIA IT Architecture model. Location model for the system on fig. A

Fig. C. UPIA IT Architecture model. Communication model for the system on fig. A

Fig. 2. Visualization of Vertical Integration

than in other methods, since in case of new or enhanced functionality, the only possible way to implement (scale of the system) would be implementing another silo. Reusing subsystems to create functionality is not possible. Integration model shown in Fig. 2A was formally represented using UML in Fig. 2B and Fig. 2C. In the presented IT architecture view, the standard localizations' elements placed in them (group of systems) and traffic routes (relations) were determined according to the UPIA standards. Meanwhile Fig. 2C presents formal representation of communication among systems described in Fig. 2A.

2.3. Horizontal integration

Horizontal Integration is often referred to as Enterprise Service Bus (ESB) due to the shape of such architecture. Implementation of the ESB allows for communication among systems to other without need for implementing additional adapters dedicated for each system. This approach allows for cutting the number of interfaces to only one per system. Each system will be connected directly to the bus using dedicated adapter. Architecture of Enterprise Service Bus is presented in (Fig. 3).

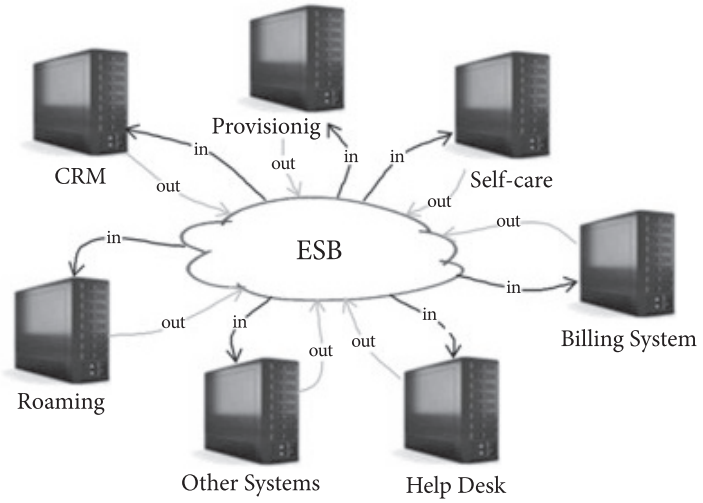

Fig. 3. Enterprise Service Bus and Legacy Systems (on sketch UML diagram)

The Enterprise Service Bus is the most flexible of all types of integration. Assuming that interfaces become stable, it is extremely easy to completely replace one system with another one. The ESB is capable of translating communication among systems, which enables cutting costs of integration and provides extreme flexibility of architecture. It is also very easy to plug new systems to the bus. If replacement/ extension of some system requires a change in interface, it is significantly less laborious because it requires modification of only one adapter.

3. SOA architecture

Service Oriented Architecture (SOA) is an architectural style that mainly put emphasis on defining services which fit to the business patterns. Services are business functionalities that are built as software components (black boxes) that can be reused for different purposes. The main purpose of SOA is to increase (optimize) collaboration between business and IT department.

The main assumptions of SOA [5]:

— SOA is for building business applications;
— SOA is a black-box component architecture;
— SOA components are loosely coupled;
— SOA components are orchestrated to link together through business process to deliver a well-defined level of service.

First of them makes an assumption that the SOA is explicitly intended for building business application. The SOA is not dedicated for building every kind of software. Despite the fact that this architectural style suits also to many various types of projects, it was designed and adapted for building business supporting applications.

As it was mentioned, one of the main SOA establishments is to hide implementation's details. The black-box approach enables us to reuse existing business applications by adding a simple adapter to them, no matter how they were built. Farther, the SOA prescribes to differentiate amount of details visible for particular users.

Term "loosely coupled" in the third assumption refers to the way how components interact with the SOA. One component passes data to another component, the second one request further to third, third to fourth and so on until request achieves the destination point. When request is synchronous, responses back to the originator through the same components which carry them during request. Each component offers a small range of simple services to other components, the emphasis is on simplicity and autonomy. The same operation could be done by tightly structured application, but the set of "loosely coupled" can be combined and recombined in myriad ways. That makes the overall IT infrastructure much more flexible.

The last but for sure not the least, is an assumption that defines how components collaborate and group to cover business requirements. The fragmentation, which is required by previous assumptions, makes a simple arrangement of components that can collectively deliver very complex services to cover requirements of the business. According to the SOA establishments, IT should provide acceptable service levels. To meet this challenge, architects should lay huge stress on quality of delivered components.

4. ESB architecture

4.1. Structure

According to the SOA assumptions, the ESB comprise components responsible for providing various functionalities at different level of abstraction. Some components are responsible for communications at physical layer's level (e.g. adapter to database), others one work as an interface in particular systems in company. The components inside the ESB as a rule are divided into sections responsible for executing business field, such as e.g. authorization of client's area which includes components executing minor tasks (e.g. downloading data to logging). Basically, we can divide the components into adapters responsible for communication (Fig. 4) with external systems (adapters) and internal ESB components fulfilling specific functionality (broker) [4] (Fig. 7).

Adapters are divided into two basic categories (Fig. 5):

— Outbound — initiated by clients, adapters that enable sending orders from system to ESB (Client Adapter);

— Inbound — initiated by ESB, adapters responsible for sending orders from ESB to domain system (Provider Adapter).

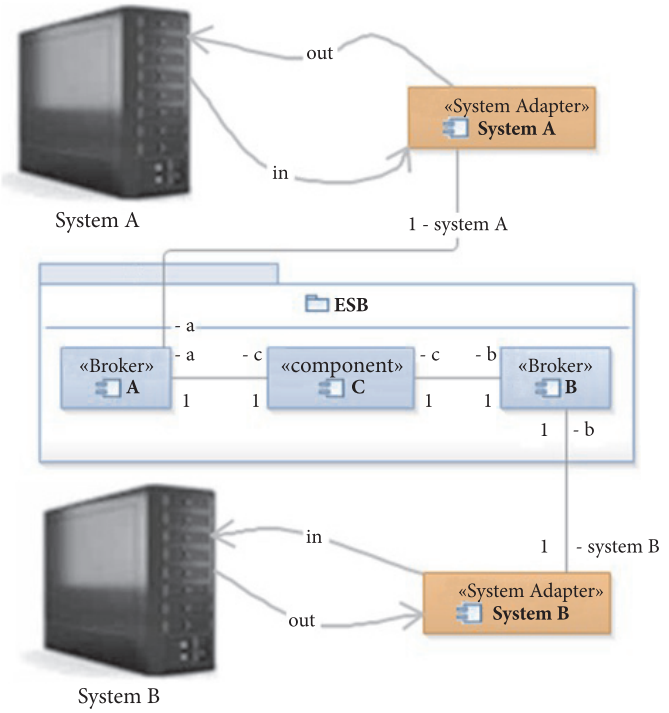

Fig. 4. Architecture of Enterprise Service Bus — Adapters and Brokers

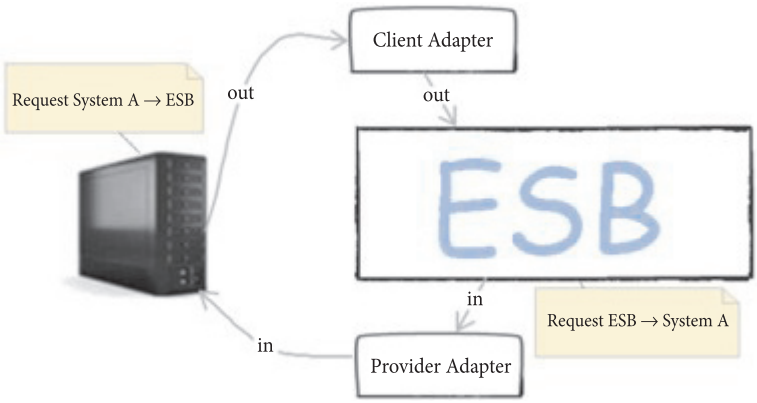

Fig. 5. Inbound and Outbound adapters

4.2. Outbound adapters

Client Adapter is a component exposing an interface, which enables sending demands and orders from domain system, and receiving answers to the demands sent earlier. Furthermore, these kinds of adapters are responsible for translating statements from legacy language (data model) into language used inside ESB (CDM).

Client Adapters applies a communicative protocol which is used by the system for which they are the adapter. At the same time, they are in charge of translating command into different pattern when the one used by the system is different from the one accepted inside the ESB. This transformation takes place twice in case of synchronous communication — first time during sending the first demand, second while receiving feedback.

The most of business operations' base on message exchange. Information about clients, addresses, services and all other are transferred from one system to another to realize business logic. Multiplicity of messages makes such management of that information difficult. To avoid mess and to make management and tracing of messages easier systems is using headers. The headers are the metadata that describe information sending as a main content. Usually, headers are designed for specified system needs and they are different among other systems. Next, duty of adapters is to translate headers from a system model to common headers used inside the ESB. Adapter must also handle a situation when domain systems do not support the headers.

Mentioned patterns were positively verified in practical implementation of ESB for the client and they allowed us to resolve problems such as:

— different data models among domain systems which were translated on the client adapters level;

— different communication patterns (more than 10 different patterns used by client);
— different headers (models, structure, multiplicity etc.), translated on the client adapters level.

4.3. Brokers

Apart from previously mentioned types of adapters, which provide implementation of simple operation, another important type of components are the Brokers — responsible for realization of complex logic queries and promoting, and delivering messages to the appropriate systems (adapters) according to the implemented rules [6] (Fig. 8). Brokers can perform very complex tasks such as:

— decomposition of demands for specific tasks;
— execution of specific tasks in a specific order;
— enrichment of demands for data from other systems;
— errors handling during the process of calling particular steps;
— service priorities in case of a large number of demands;
— queuing incoming demands.

Brokers, also called Composite Services, implemented in corporation can be very complex services executing above mentioned tasks at once or can be minor processes executing only part of the presented functionalities.

A characteristic feature of Composite Services is its placement — inside the ESB and thus it is restricted to legacy systems through adapters. Thanks to this architecture, Composite Services use purely and simply Canonical Data Model both — the input and output of components (Fig. 6). This allows us, according to the SOA assumptions, to realize purely and simply the tasks assigned to specific components.

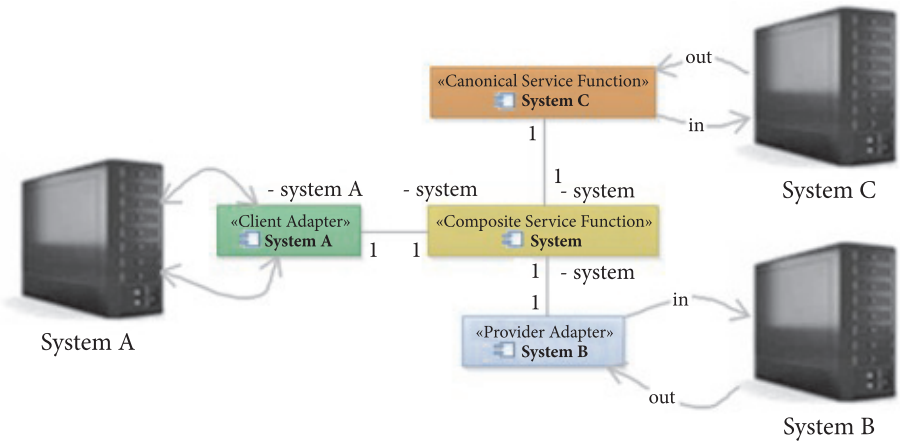

Fig. 6. Composite service separation

4.4. Inbound adapters

Provider Adapters are used when the ESB needs to obtain information from the system or to produce other services provided by the specific server. Division of adapters into inbound and outbound ones, results from both — established the SOA (division of components into small ones, performing a single function) and from the fact that although the components implement similar functionality (providing communication of the ESB — system) produce them in a different way.

Similar to Client Adapters, the Provider Adapters have to adjust to the system's communication pattern with which they communicate. Furthermore, they are in charge of transcoding data (headers, contents, errors) and the communication model into a model corresponding to a specific system as well.

However, the adapters of this kind as opposed to Client Adapters need to implement error handling. In this case, only translation of the codes is not sufficient. The component has to check the additional conditions, e.g., the system can return empty list of results, which from the technical point of view would be the correct execution of the request, however, further operation on the empty set may cause errors inside the ESB. In such a case, the Provider Adapter should detect this situation and return the information concerning business error to the ESB (empty list data).

The main difference between the adapter inbound and outbound type, is the place which they occupy in order of execution of a specific workflow. Because the Client Adapter is at the beginning of the flow, in case of sending incorrect data from the system, it is enough for it to return information concerning error to the calling demand system and thus it completes the process. The demand will be stopped at the very beginning and no steps will be taken until the system sends the correct message. As far as Provider Adapter is concerned, validation of data returned from the system is insufficient. In this case, an error has to be recognized and appropriate message about its appearance should be reported to the ESB so that, the controlling component could take adequate steps.

In case the component only returned errors to the domain system and thus finished operation, it would cause undesirable effects. For example, if such an error appeared during launching new service, at the stage of physical activation of service, it could appear that although service was not activated (because of an error), the customer would be charged because charging process would run properly (even though the service would not run correctly owing to the error). Moreover, we can imagine a situation when the person activating some service was sending the demand several times, because of lack of feedback whether the service had been activated or not, so the client would be charged repeatedly although the service would not be activated.

Described components are the basic elements of the bus. Using them allows for building very complex structures, responsible for realization of difficult functionalities. However, in case of large system integrations (which usually expose many operations),

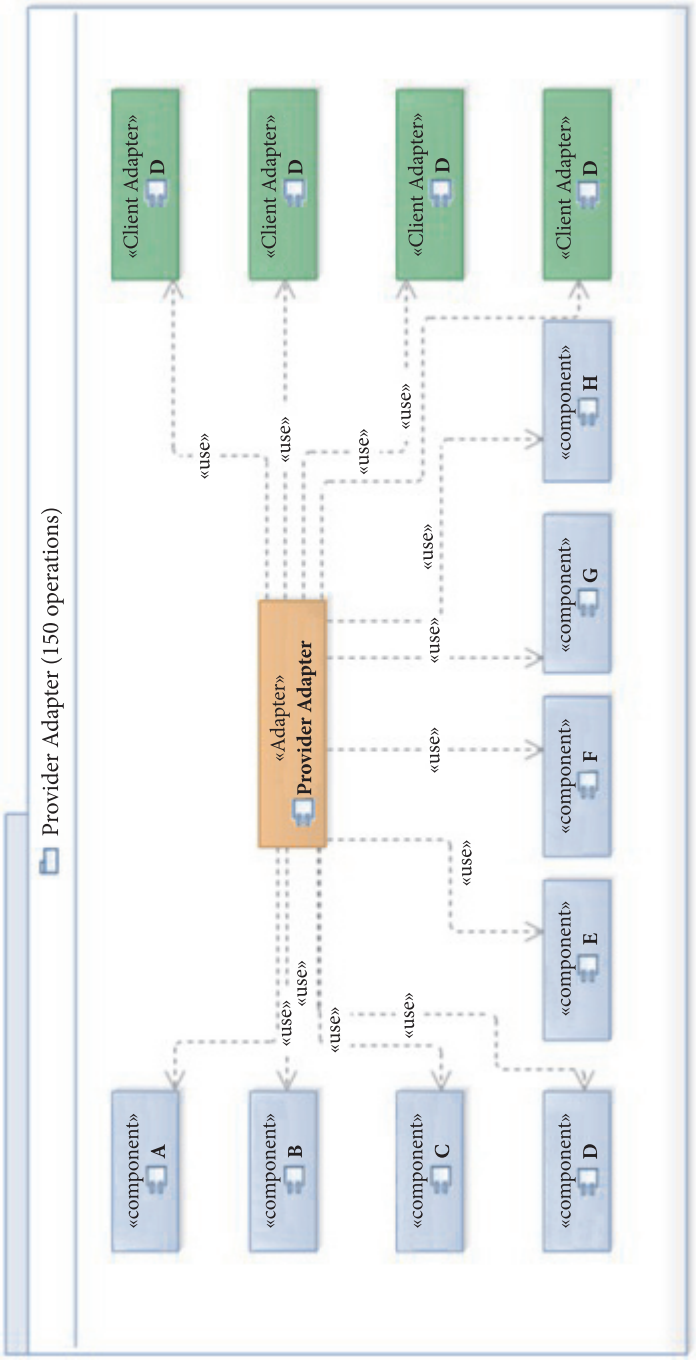

Fig. 7. Adapter overload

the problem of adapters' overload appears (Fig. 7). Similar problem occurred while implementation of ESB for mentioned telco client. In the first stages, when adapter was responsible for small amount of processes, storing all the operations in one adapter was gratifying approach. However, while growing and increasing of the bus, the problem of adapter overload has appeared which has caused negative effects.

In case of adapter responsible for more than 100 operations, it becomes bottleneck of the whole system. Even the smallest breakdown of some part of adapter can cause the downtime of the whole system. Analogical situation appears in case of adapter enhancement — partly upgrading (of certain area) of adapter causes exclusion of the whole adapter. In practice, it means that the whole system is temporary unavailable for clients.

The next problem related to the overload of the adapter can be performance. Such a situation can concern both, capacity of a single adapter (which can be forced to serve few/tens operations per second) and data transfer (caused by many requests to the adapter generated by other components). As a result, some of them may not receive response in the assumed period. Such a situation may require the change of a communication pattern (synchronous to asynchronous) or implementation of queue mechanism.

Since the crucial part of majority is availability on the 99% level, one of the most important activities to be taken is recognition and elimination of bottlenecks among enterprise. The simplest way to achieve that seems to be adding new hardware in order to increase performance. However, it generates additional costs. What is more, eventually, the system reaches the critical point where hardware incensement is not possible or efficient. Taking that into account, the decision was to formalize research problem as: "ESB bus design optimization in the terms of resources". The aim was to resolve the problem without increasing hardware architecture.

As a result of research and measurement taken on Prove of Concept components it was found that the best way to increase performance and use hardware resources in more efficient way was to apply a process of canonization and aggregation. During the research, two different approaches were taken into consideration — service aggregation by:

- domain similarity;
- monitoring of ESB load, simulation of ESB traffic using queue systems (future solution).

Since final decision belongs to stakeholders, it was decided to deploy first solution, however, it needs to be noticed that the second approach could be even more efficient and will be the subject of future research. Implementation was done using TIBCO Active Matrix BusinessWorks software which, according to Forrester Research, is one of the leaders in SOA/ESB tools [7].

As a result, logic was distracted for the major amount of components — canonical and provider adapters. Canonical adapters are responsible for some business area

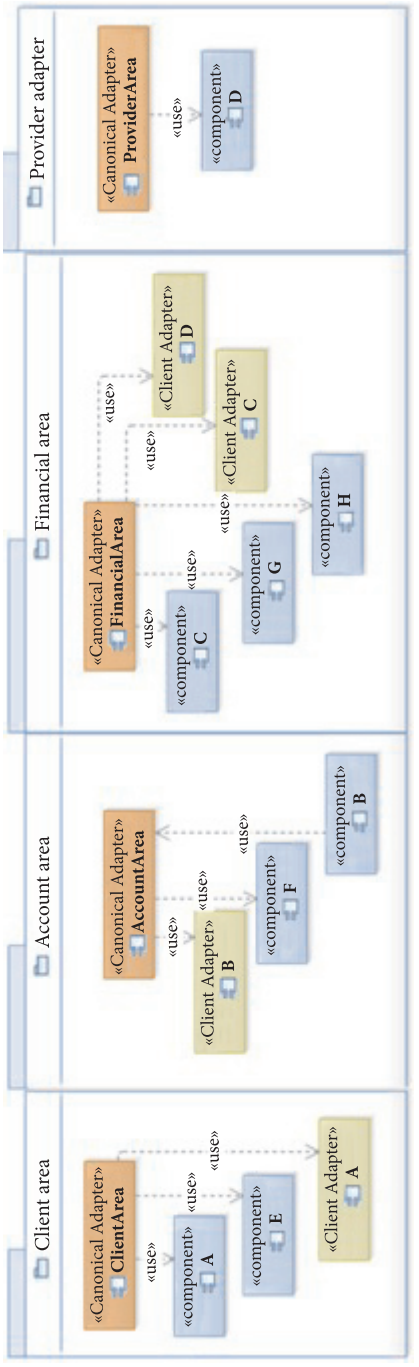

Fig. 8. Canonization

and they contain operations related to it. In case of the client's implementation it was decided to split adapter for CRM system to canonical adapters responsible for:

— operations related to the client object;
— operations related to the client's account object;
— operations related to the financial transactions;
— operations related to the client's interaction;
— operations related to the self-care interaction;

This process allows us to exclude some functional areas (Fig. 8) (containing common features) and its aggregation. This division reflects real business flows.

When the operation generated in domain system is characteristic for only one flow (is used by only one component), it is recommended to create private adapter in the form of provider adapter (Fig. 9). When the operation, generated by adapter, is used by more internal flows (it communicates with more than one components), transforming into canonical adapter is suggested.

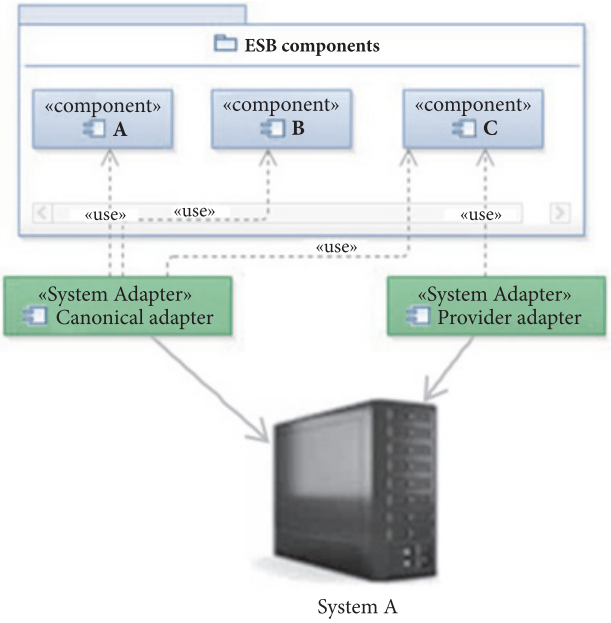

Fig. 9. Canonical and Provider Adapters

Canonization of flow and split of adapter into canonical adapters [3] causes many advantages. The first one and the most visible is growth of clarity code and gaining structure responding to the business model (certain flows reflect business areas). Systems related to billing area will contact only dedicated adapters responsible for these areas, without loading other components.

Decomposition allows also decreasing the risk connected with enhancement implementation. In case of error caused by update, accident impacts considerably smaller amount of flows. Similar situation takes place in case of failure — temporary downtime of one component does not block the whole environment.

Apart from decrease in component load by reductions of invocations, division allows us to increase capacity by assignment of different amount of resources. In case of client environment, adapter served the processes which were performed with different frequency. The rarest ones were executed one per a few days while the most frequent — hundreds per hour. In order to imagine the canonization advantages (Fig. 10), we can see exemplary amount of operations (from different areas) executed per one hour.

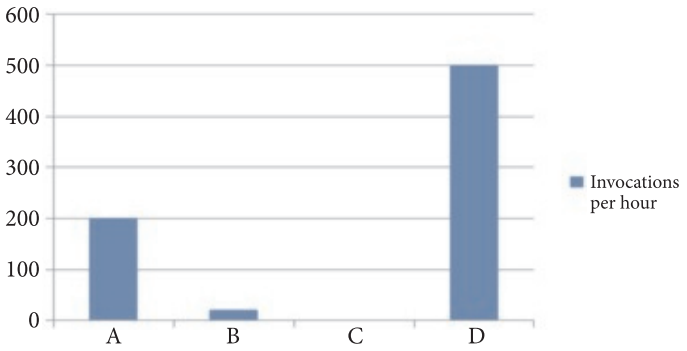

Fig. 10. Approximate invocations/hour

A operation was invocated approximately 200 times per hour, B — 20 times, C — 2 times, and D — about 500 times. In such a situation, the adapter had to be run on the machine which resources allowing to cover needs of D area, while the areas B and C used only few percentages of available resources. After split, like it was shown in Fig. 8, it was possible to run the dedicated adapter for D area on independent machine which has greater resources while the rest ones could be run on machine, proportional to a number of processes occurrences.

5. Verification of solution

Measurement of architectural solution quality is a complex problem. During justifications of quality, different measures can be used; both quality and quantity. Among quantity solutions it is proposed to use the following measures:

— measure of a connection degree — understood as an average number of combinations which are used to message exchange and control among measured element and other elements of a system;

— measure of cohesion degree — understood as a level of internal element consistent.

Using such defined measures shows that after canonization process, canonical adapters are characterized even 10 times less level of connection degree depends on granularity division. In some situations (dedicated adapter), connections degree can be even 100 times less.

Using a measure of cohesion degree, understood as an X/Y ratio, where X are connections responsible for realization of business logic from certain area, Y — are all connections inside adapter, we receive similar results. From 10 to 100 times more than in case of single adapter.

The proposed rule of canonical adapters' creation, which consists of aggregation of the elements, on the basis of belonging to the functional areas of the system (X/Y measure), was verified in real environment. However, it is one of many possibilities.

In further work, it is planned to improve the proposed rules by using queuing systems to predict the use of adapters and their components. In this method, the research team assumed that a rule of aggregation on the basis of belonging to the functional areas will be defining only an "initial structure". Following iteration will made evolution of the structure until the point which will be optimal for "network traffic" on the assumed characteristics.

Currently, the team is expanding the functions of dedicated monitor for collecting statistics on calls (use) adapters, as well as their components and resources' demands (taking into account distinction between day and night).

This monitor will allow us to collect network traffic characteristics, which will then be mapped in queuing systems. Subsequently, we plan to determine the adaptive algorithm limits adapter's structures with minimal resource demands.

6. Conclusions

As presented examples describe, when implementations of huge structure (more than 40 components) or components serving many operations (approximately more than 50) standard patterns are not enough efficient. At the mentioned or even higher level of complexity, canonization process becomes necessary to avoid negative results of components overloading. Furthermore, practical experience shows, that canonizations is no longer just a way of improvement but natural consequence of platform growth in service oriented companies.

Nevertheless, it needs to be noticed that implemented solution carries potential threats. Process of canonization helps to eliminate bottlenecks, however, we can assume that due to constant incensement of operations amount, eventually the new bottlenecks will appear. Theirs elimination by constant canonization can create

distributed network of small components which will be impossible to manage. Eventually, it will lead to exactly opposite results as it was assumed according to SOA paradigms (single point of access). As a result, in such a situation it is recommended to remodel or redesign architecture instead of applying the proposed solution.

Received November 7 2012, revised Fabruary 2013.

REFERENCES

[1] Comm server, www.comsvr.comm [online] [25.12.2011] http://www.commsvr.com/howitworks/architecture/integration.aspx

[2] Elliott, Michael [online] [26.12.2011] http://www.e-ditionsbyfry.com/olive/ode/sci/default.aspx?href=sci%2f2011%2f05%2f01&pageno=19&entity=ar01902&view=enti

[3] B. Gold-Bernstein, W. Ruh, *Enterprise integration: the essential guide to integration solutions*, 2005.

[4] M. Ross, http://blogs.mulesoft.org [online] [27.12.2011] http://blogs.mulesoft.org/esb-or-not-to-esb-revisited-part-1/

[5] Ibm. Ibm: developerworks [online] [25.12.2012] http://www.ibm.com/developerworks/webservices/library/ws-esbscen3/

[6] Arsanjani, Kerrie Holley, *dr. Ali, 100 soa questions: asked and answered*, November 22, 2010.

[7] Forrester Research, The Forrester wave™: enterprise service bus, q2 2011, April 25, 2011, http://www.progress.com/docs/campaign/analyst/2011_forrester-esb-wave.pdf

D. ADACH, W. DĄBROWSKI, A. STASIAK

Integracja systemów przez integrującą magistralę usług w architekturze SOA z użyciem wzorców projektowych

Streszczenie. W artykule opisano metodę określania wzorców struktur adapterów dla systemów wykorzystujących magistralę ESB. W zaproponowanej metodzie, poszukiwane podczas projektowania adapterów wzorcowe struktury, są wyznaczane w wyniku odpowiedniego podziału adapterów dedykowanych. Utworzone adaptery są silnie związane wewnętrznie, ponieważ po procesie kanonizacji umożliwiają komunikację tylko z procesami tej samej domeny (obszaru funkcjonalnego). Dodatkowo założono, że wyznaczone wzorce powinny prowadzić do eliminacji punktów krytycznych i wąskich gardeł budowanego systemu, oraz wzrostu wydajności komunikacji i efektywności zarządzania zasobami. Metodę przebadano w środowisku Tibco ESB, podczas prac nad rozwiązaniem dla operatora telekomunikacyjnego.

Słowa kluczowe: integracja systemów (SI), integracja aplikacji korporacyjnych (EAI), korporacyjna magistrala usług (ESB), wzorce projektowe dla korporacyjnej magistrali usług, kanoniczne adaptery magistrali usług ESB, architektura zorientowana na usługi (SOA), zunifikowany język modelowania (UML)