The concept of server allocation to components in SOA architecture

A. WOŹNIAK
adrian.p.wozniak@gmail.com

Institute of Computer and Information Systems
Faculty of Cybernetics, Military University of Technology
Kaliskiego Str. 2, 00-908 Warsaw

This paper describes the concept of the allocation method. By using this method we can optimize allocation of servers to SOA components which facilitates business processes in the organization. The method is based on taking an acceptable solution and then simulating it iteratively. Iterative simulation is used to find better allocation. Simulation helps in estimating efficiency and supports algorithm used to find a better solution. The main aim of the method is to reduce the time needed for business process realisation. This work shows the source of the problem and provides a short introduction about Service Oriented Architecture. It describes how to build components in SOA and what role they play. This paper also describes other methods which were used to solve the problem, in particular static and dynamic methods that allocate components to servers in SOA. A new method is described that also solves the problem but in a new way. This paper shows the algorithm of the new method as a whole and provides details about each step and basics of how a simulation model is constructed.

Key words: SOA, optimisation of components allocation, digital simulation.

1. Introduction

The management of business processes in a company consists of, inter alia, optimisation i.e. increasing efficiency. The efficiency can be increased by reducing the time needed to complete business process. The time of business process realisation can be shortened by:
- reorganizing processes e.g. deleting redundant tasks or making them to be realized in parallel,
- increasing resources of organization. It is effective when resources are a bottleneck of business processes. Bottlenecks can be located by e.g. business process simulation,
- shortening time needed to realize tasks in business process. It happens usually by automation, especially by computer systems that can do tasks instead of human.

A method presented in the article concerns last two ways of optimizing business processes.

With evolution of software engineering, architects of systems use abstracts on a higher level. At first, software was constructed by using function libraries. Then, there were objects and later, component structure of software. The next step of this evolution is Service Oriented Architecture (SOA). According to SOA, the system should be divided into independent modules. Division should be made by business criteria instead of technical ones. Each module’s job is to support business processes of organization. Every module has its area of responsibility and it provides services to other modules. Every service provided by the system has to have business meaning and, has to be a step in the business process. Furthermore, input and output parameters for services also have to be understandable from the business point of view. Purely technical parameters are not allowed.

In SOA there is a direct relationship between steps in the business process and services used to realise them. In Business Process Model and Notation (BPMN) services invoked are represented as “Service task”. Usually before service invocation there is “User task” in the process which represents using functionality of the system by user (usually through GUI). After “User task” system realises part of the process automatically by invoking services in order defined in the process.

There are two ways to compose services: orchestration and choreography. The first one uses central and universal module which directs all invocations of services in accordance with business process that is defined within that module. Implementation of concept of central module is Enterprise Service Bus (ESB) with Business Processes Engine. ESB is a broker between modules in invoking all services within system. Business Process Engine is a place where all business processes are defined and which is responsible for realizing business
processes according to that definition. It also monitors realization of business processes. The name “orchestration” comes from association with a conductor and an orchestra. The conductor is ESB with Business Process Engine and the orchestra are modules. On the other hand, there is choreography where each module decides which service should be invoked next. To make this possible business processes should be implemented within modules. Using choreography it is easier to implement software but modules are less reusable. ESB is also used within choreography for its many benefits.

![Diagram of SOA Architecture]

Fig. 1. Collaborations in SOA Source [16]

Every module constitutes separate and independent software. Each one can be implemented in different technology and requires different runtime environment. It is important, however, that services should be provided according to SOA standards. So it should be possible for every service to be invoked by other modules, ESB or Business Process Engine. Each module plays at least one of three roles: Service provider, Service client or Service registry (Figure 1). Service providers are modules that realize services. Service clients are modules that need to invoke services to work properly. Service registry is a broker that contains information about services within the system. ESB is an example of such Service registry which is used in all service invocations within the system. Using ESB gives more control over the system and provides more security. Every provider must publish services on ESB. It is ESB that provides services to other modules. When a client wants to invoke service he sends a request to ESB which is responsible for handling requests and communication with a provider. ESB serves as:

- Translator between modules. It is used when providers and clients use different interfaces to communicate.
- Loadbalancer – which is used to govern load on servers. It gives possibility to connect many providers of the same service or even many instances of the same module.

The fact that there are many service providers and that they operate on different servers can be hidden from the client. From the client’s point of view, the request is sent to ESB which is responsible for sending it to the best service provider or providers.

- Security layer to control traffic between modules.
- System monitor that gathers information about processes realized in the system.

SOA allows a direct relationship between steps of business process and services invoked in the system. Without such relationship optimisation of allocation of servers to components could be done only from the point of view of the system (not business). For example, such optimisation can mean a reduction in: service response time, system average response time, probability of service realisation within a given time, etc. Locally optimal service efficiency does not mean global optimal work of business processes. Thanks to SOA optimisation can be done in a new dimension. In SOA, the allocation of servers to components can be done from the point of view of business processes. In SOA we can calculate the impact of allocation changes on business characteristics (not only system characteristics).

2. Available solutions

In the literature you can find publications presenting methods that optimise allocation of servers to software components made in SOA. Those methods can be divided into two groups: static and dynamic.

In dynamic methods, allocation of hardware resources to software components is changing dynamically in short periods of time. Allocation is dependent on current load of the system. In dynamic methods load is defined by running instances of business processes and step on which they currently are. Using that knowledge about system load and about processes we can predict which services will be invoked within next time. Therefore, we can predict system load. Such a prognosis is the basis for calculating optimal allocation. In [1], example of such dynamic method is presented. The article shows an example of the dynamic method where summary realisation time of current running processes is minimised. Article [2] presents a method which is focused on optimisation of Quality of Service (QoS). In [2] optimisation of QoS is defined as maximisation of probability that complex services are realised in set time.
Complex service in this case consists of service invocations in order defined by business process, in fact it is a part of business process. So in this method we have to define time in which complex service should be realised. Complex service is considered as realised when response time was lower than previously set time. In both dynamic allocation methods many simplifications have been adopted. In the method presented in [1] such elements as load and time delay from changing allocation are completely omitted. Load generated by runtime environments is also omitted. In method [2] only delay from changing resources allocation has been included.

The advantage of dynamic methods is the ability to react on dynamically changing system load. This advantage is at a cost of building a mechanism that monitors and switches resources between components. Another cost is a bigger load of servers because of sustaining that mechanism. Thanks to that kind of optimisation, current running processes should be realised faster but it will be less prepared for processes coming next. For such processes system will adopt with delay. Such dynamic methods are applicable in supporting business processes in which intensity of invocation starting events is changing very quickly and significantly.

In SOA many different technologies can be used and also current solutions can be adopted. Therefore, dynamic methods sometimes cannot be used because not all technologies can support changing the resource allocation so quickly.

The second category of allocation methods are static methods in which one optimal solution for the system should be found, and will not change with time. Static methods often use mathematical models of components and resources. Such mathematical models are often build with the use of queuing networks and Markov chains. The example of such a method is [3] in which authors present a generic method for allocation optimisation in SOA and CORBA architectures. Only one parameter that is changed during optimisation is the number of virtual machines. In this method virtual servers are identical by definition. Moreover, we can multiply all modules on the unlimited number of virtual machines. Methods with the same basis of modus operandi are presented in examples [4] and [5].

Advantages of static methods in comparison with dynamic are the lack of additional load on servers and the lack of need to build additional mechanisms to monitor, control and change allocation of resources to components. By using estimation of process invocation intensity, systems with this kind of allocation methods are better prepared for upcoming tasks. It is because dynamic methods only react for currently running processes. Disadvantages of such solution are the need to predict and estimate intensity of business process invocations before deploying the system and lower effectiveness in environments where intensity of such invocations changes significantly.

In both kinds of methods characteristics of system are deterministic. In reality those characteristics are random. In particular processes invocations are usually random. Therefore, the results of optimisation with presented methods might encounter problems to meet the standards of QoS.

3. Optimisation Method of allocation of hardware resources to software components made in SOA

The proposed method has some features of both static and dynamic methods. It also uses dependencies between business processes and invoked services that were presented earlier. The knowledge of business processes and intensity with which they are invoked lets us predict load on components. Using knowledge about dependencies between processes and services, and about load on components we can find efficient allocation, just like in static methods. On the other hand, some flexibility of dynamic methods has been achieved. Within the method, there are defined parameters that should be monitored by the system. Every monitored parameter has assigned threshold. When value of parameter exceeds the threshold another allocation iteration should be done, just like in dynamic methods. But presented method does not need implementation of mechanism for quick allocation changes because they are not so dynamic. Therefore, the method does not require changing allocation during runtime. Change of allocation can be done during deployment so the method does not generate additional load on the system. To evaluate allocation, simulation is used. Thus, the method does not impose such strong restrictions like static methods do. Moreover, using simulation allows modelling randomness of system characteristics, so it is possible to impose restrictions that are related to QoS (e.g. process cannot be realised for longer than 1 second with probability of 99.5%).
Proposed method consists of two parts:

a) finding suboptimal allocation of resources to components,
b) monitoring a solution to check if condition of simulation changed significantly (if yes, algorithm returns to the first step).

The part of the method that is designated to optimise allocation consist of four steps:

a) preparing one or more initial allocation of resources to components,
b) simulating SOA system model to evaluate quality of solutions obtained so far,
c) deciding whether continue optimisation process or not,
d) improving solution.

Steps 2 to 4 are performed iteratively until algorithm stops as shown in Figure 2.

![Optimisation algorithm of allocation of resources to components](image)

3.1. Simulation model

Simulation model consists of four layers that overlap. They are:

- servers and network,
- runtime environments,
- components and services,
- business processes.

Every consecutive layer is dependent on the previous one (Figure 3).

3.1.1. Servers and network

First layer is made of hardware that is used to run software. It consist of servers, network nodes, and their capacities. In this layer we define:

a) servers described with attributes:
   - processor power – the number of operations per second,
   - memory size – the number of megabytes of available memory,
   - input capacity – the capacity with which data can be transferred to server (mb/s),
   - output capacity – the capacity with which data can be transferred out of server (mb/s).

b) network nodes (generally routers) with attributes:
   - input capacity,
   - output capacity.

c) connections between servers and network nodes. Connections are presented as binary matrix.

3.1.2. Runtime environments

The second layer is created by runtime environments installed on servers. Those environments are necessary to run components because all components require some middleware to be run on (e.g. web applications build in java need middleware like Glassfish on Linux which is runtime environment). Every environment running on system generates additional load. In this layer we define:

a) Runtime environments described with attributes:
   - processor load – the number of operations per second that are needed for its proper operation,
   - memory load – the number of megabytes of server memory needed for its proper operation.

b) Allocation of runtime environments to servers. It is a binary matrix that shows to every server, which environment is installed.
This allocation is variable, which is very important for optimisation of allocation servers to components. Allocation of runtime environments to servers must be done in such a way that components allocated to servers can work properly. Therefore allocation of runtime environments is a consequence of the allocation of components to servers.

3.1.3. Components and services

In this layer components are defined. A list of provided services is assigned to each component. Those services are used to realise steps in business processes. In this layer we define objects:

a) Components described with attributes:
   - processor load – the number of operations per second that are needed for components proper operation (load generated by invoking services is not included),
   - memory load – the number of megabytes of server memory needed for component’s proper operation (load generated by invoking services is not included also),
   - runtime environments – the list of runtime environments on which component can operate properly,
   - restrictions on the servers – the list of servers on which component cannot be installed because of organisational reasons. It generally concerns situations where organisation or division of organisation is responsible for realising some kind of operations or responsible for managing data contained in a module, so it has to have full control over component.

b) Services described with attributes:
   - processor usage – the number of operations required to realise service,
   - memory usage – the number of megabytes used during realisation of service,
   - input data size – the number of megabytes that has to be transferred from service client to service provider to realise service,
   - service providers – the list of components that are able to provide service.

c) Allocation of servers to components – it is decision variable which significantly influences realisation time of business processes. It also has a great impact on allocation of runtime environments to servers, which must be done in such a way that all components can operate on their assigned servers. One server can run many components if its resources allow it.

![Fig. 3. Layers in simulation model](image-url)
3.1.4. Business Processes

Within this layer business processes are defined but only to a level necessary to simulate them and optimise allocation. Business processes consist of activities that can be realised by people or the system. Decisions within processes are represented only in a statistical way i.e. only possible transitions and their probability are important. Within this layer we define objects:

a) Business Processes with attributes:
   - invoke intensity – the probability of invoking business process within unit of time,
   - tasks – the sequence of steps that show order of invoking services required to realise process. If process has gateways then they are described by probabilities of choosing each of them.

Method does not include tasks that are realised outside the system. It is because within the system we have no influence on how long those tasks will be realised. The aim of the method is to minimise realisation time of business processes. If we compare two processes with the same time needed to realise steps outside the system, then the only difference will be within steps realised by the system. Therefore, we can skip steps outside the system without any loss in quality of allocation of servers to components.

3.2. Evaluation and improvement of allocation

Every optimisation algorithm requires an objective function. Within the proposed method we optimise realisation time of business process. By realisation time we mean time between onset within the system and ending realisation (which means that we take into account time of waiting for resources). If all business processes are equally important then we minimise total realisation time of all invocations of business processes. If some business processes are more important than others then we minimise weighted total realisation time of all invocations of business processes. Weights are assigned to all business processes and are estimated from e.g. value of the business process for organisation, then if the business process has two times larger weight it means that it gives two times larger income for organisation. Evaluation is done on the basis of simulation of model presented in chapter 3.1.

The allocation of servers to components improvement is done by using genetic algorithm. If M is the number of servers and N is the number of components then:

Matrix X such that:

\[ X = [x_{ij}]_{M \times N}, \]

\[ x_{ij} = 1, \text{ in case where server number } i \text{ is allocated to component number } j, \]

\[ x_{ij} = 0 \text{ otherwise, } \]

is current propose of allocation. Such Matrix we call phenotype which will be further used in genetic algorithm. At the beginning of algorithm, many phenotypes are generated.

Further steps are realised according to definition of genetic algorithm. Therefore, for each phenotype evaluation is associated based on simulation. Next step is the selection of best solution candidates and generating new set of phenotypes by using operators of crossover and mutation. A new set of solution candidates is evaluated. Crossover and mutation operators will be developed during further work on the method.

To avoid the situation where generated allocations are unable to realise some processes, penalties are used. For each process invocation that cannot be realised during simulation, a penalty point is assigned and multiplied by the weight of business process. Best solution candidates are those with the lowest amount of penalty points. Among solutions with no penalty points, best phenotypes are those with the lowest weighted total realisation time of all invocations of business processes.

Fig. 4. Genetic algorithm of resource allocation optimisation
According to approach selected to model SOA architecture, expected results of allocation should be totally different. For system modelled using orchestration we expect that on server with the highest transfer capacity, algorithm will allocate ESB with Business Process Engine. In such a case optimal structure will take shape of star where ESB is central element. If choreography is used to model system then we should expect distributed network structure. If both approaches are used, especially when many ESB are used then we expect mixed structure.

Using simulation to evaluate solution allows to estimate many other characteristics of system. It especially includes characteristics related to the quality of system services such as:
- resistance to anomalies (e.g. large amount of service invocations in short time),
- probability of failure to realise service,
- probability of realising service in a longer than set time.

One of allocation restriction can be meeting condition of having better characteristics then previously set.

3.3. Monitoring changes

Business processes together with IT systems are subject to constant change. Monitoring of business processes and systems is included in the proposed method to maintain optimal allocation in long term. Some part of simulation model characteristics is changing discretely while the other is changing continuously. For example, a decision about changing business process is done discretely. On the other hand, there is e.g. intensity of business process invocation which is changing continuously. For both kinds of characteristics the threshold should be set. After crossing the threshold, the next step is finding new allocation and for this allocation setting new thresholds for parameters. In SOA Business Process Engines are used to store, run and manage business processes. Business Process Engines also has the ability to monitor business process characteristics such as intensity of process invocation.

4. Conclusion

Optimisation of allocation of hardware resources to components is rarely used in practice. There are at least two reasons for that. Firstly, optimisation of such allocation is not part of IT projects so it would generate unnecessary cost from project manager’s point of view. There is little awareness on both sides (business and IT) about optimising business processes. Secondly, methods that already exist differ from reality because of simplifications used in methods. Thanks to the proposed method based on simulation more systems can be modelled realistically and for those better allocation solutions can be proposed. However, the proposed method also has some limitations in practical application. It requires knowledge about not only business processes but also intensity of their invocation. Therefore, it is usable in organisation that is performing business process simulation or is able to do it. It requires to arrange more input data about system than other methods (it requires e.g. runtime environments characteristics). It also does not impose so many restrictions like static methods. On the other hand, it is better prepared for upcoming business processes than dynamic methods. Moreover, it does not require additional mechanisms to switch resources between components.

The choice of optimisation method always should be based on the level of knowledge about supported processes and the ability to meet the requirements of static methods. Limited knowledge about processes, or a situation where they are changing so fast that their invocations are hard to predict, would suggest using the dynamic method. If the system can satisfy the constraints of the static method then it seems to be best to choose the static method. For systems where business process invocation is predictable (we have estimations on probability distribution) and cannot meet restrictions of the static method (most of cases) than proposed method is recommended.

5. Bibliography

Koncepcja metody przydziału komponentów do serwerów w architekturze SOA

A. WOŹNIAK

W artykule przedstawiono koncepcję metody optymalizacji przydziału komponentów funkcjonalnych do zasobów sprzętowych w systemach zbudowanych w oparciu o architekturę SOA. Metoda ta bazuje na przyjęciu dopuszczalnego rozwiązania, a następnie na wykonywanej iteracyjnie symulacji i w oparciu o jej wyniki poprawianym jest przydział. Symulacja pomaga w oszacowaniu efektywności aktualnego rozwiązania i wspiera algorytm poprawiania rozwiązania w przygotowywaniu zmian przydziału. Podstawowym celem metody jest skrócenie czasu realizacji procesów biznesowych.

W pracy przedstawiono źródło problemu oraz krótkie wprowadzenie o tym, czym jest architektura SOA. Opisano, czym są komponenty budowane w architekturze SOA oraz jaką pełnią rolę. Omówiono prace, jakie zostały zrealizowane w ramach tego zagadnienia, a w szczególności podejście statyczne i dynamiczne do optymalizacji przydziału komponentów w architekturze SOA. Zawarto propozycję nowej metody optymalizacji, która rozwiązuje zdefiniowany wcześniej problem. Opisano algorytm optymalizacji, a następnie bardziej szczegółowo kolejne kroki jego realizacji, pokazana też została konstrukcja modelu symulacyjnego.

Słowa kluczowe: SOA, optymalizacja przydziału komponentów, symulacja komputerowa.