Model-Driven QoS-aware Approach for the Sensor Network

Assel Akzhalova
Department of Computer Engineering, Kazakh-British Technical University, Tole bi, Almaty, Kazakhstan
a.akzhalova@kbtu.kz

Keywords: Model-Driven, QoS, Optimal, Policy.

Abstract: The key idea of this article is to apply Model-Driven QoS-aware approach to the wireless sensor network that are controlled by the network of ”smart” controllers. The sensors are connected via SPI interface to the controllers installed at the oil wells in order to provide smooth data collection and transmission to meet quality of service requirements. One of the new approaches to implement decentralized adaptation technique, in particular, self-organizing transmission system. The self-organizing system control is entirely dependent on the decision taken at the local level, i.e at the level of system components. However, it is hard to reach global attainability of Quality of Service (QoS) requirements at run-time. In our work, we propose Model-Driven Architecture, the meta-model and its semantics, as the basis for an adaptation framework. The adaptation is realized as an automatic transformation through the policy generation.

1 INTRODUCTION

Service-oriented architecture (SOA) is currently one of the most sophisticated technologies used in modern e-business. The SOA principles illustrate that an integration of heterogeneous business resources such as legacy systems, business partner applications, and department-specific solutions are still emergent topics and there is a need in developing less costly, reusable and interoperable SOA solutions. In practice, there are a lot of problems that make SOA solutions complicated:

Large distributed systems often require huge efforts to deal with legacy in case of incorporating new services into the existing system in order to meet new business requirements. This problem requires research in service composition concerns including developing service description specifications, service discovery, optimal service selection, and binding protocols.

The services might be deployed on different platforms that causes problems in data exchange and increases investments in maintainability of the new system. The solution of this issue suggests elaboration of service functionality implementation providing autonomy and loose-coupling.

Different owners oblige to deal with negotiations and contracts between partners and, therefore, there is a question on how to provide Service Level Agreement (SLA) between participants.

All above are crucial for SOA systems to meet QOS requirements such as security, reliability and performance, especially at runtime.

The one of the ways of resolving the problem is Enterprise Service Bus (ESB) which breaks up the integration logic into easily manageable independent entities. However, the ESBs presented in the market are still have open disputes on the three main topics which are essential requirements stipulated by e-business:

- The SOA systems integration of heterogeneous applications across disparate systems in a flexible and less cost fashion (Martin Keen, 2004).
- Open standards as they are base for successful interoperability across heterogeneous systems (Martin Keen, 2004).
- The SOA systems automatic adaptation in order to meet required constraints and agreements. systems.

An automatic adaptation of SOA can be considered as an automatic service selection. The automatic service selection can be defined by policies that describe contracts between participants containing conditions and actions including penalties in case if the condition will not be met. For example, for the complex service based systems service providers may demand for various charging policies such as payment per resource usage, payment on lifetime services, and also specify available throughput and other con-

Akzhalova A.
Model-Driven QoS-aware Approach for the Sensor Network.
DOI: 10.5220/0005424901130122
In *Proceedings of the Fourth International Symposium on Business Modeling and Software Design (BMSD 2014)*, pages 113-122
ISBN: 978-989-758-032-1

straints. These policies generate Service Level Agreements (SLAs) legally binding contracts that sets constraints on different QoS metrics.

An example of business process that illustrates service selection via policy mechanism is shown in Figure 1. Figure 1 demonstrates run-time system of the oil reservoir automation control framework.

The oil reservoir automation system consists of a number of allocated set of sensors embedded to each oil well equipment that measure various parameters. Each oil well has one intelligent controller for collection and transmission of data measured by sensors. In case of loss of communications with the server, it is necessary to continuously transfer data using an alternative method or, in other words, to guarantee certain level of reliability.

We introduce following assumptions for the node in the sensor network and its attributes:

- Each node has the same logic, transmission and storage of data.
- Each node has the same technical specifications including processor that can perform data (Out) and low level computations and, consequently, received data packed and queued while waiting their processing.
- Each node receives data from several sensors and from neighboring nodes.
- Each node knows how many neighbors (sources) surrounded it and it is able to locally measure the number of packets received from neighbors. Each source has a fair bit of buffer space in a queue. That is each source has own ”channel” not available to other sources. Therefore, this infrastructure is typical example of queuing network.
- The ratio Tp / Ts (passive / sleep timers) determines the cost of energy and the system response to the dynamic changes.

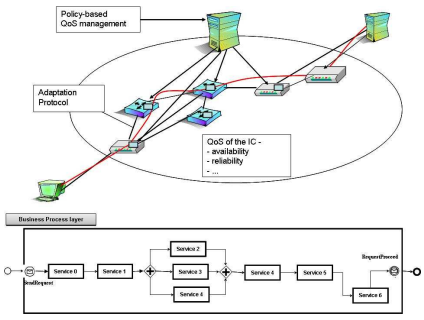

Figure 1: Business process modeling of QoS-aware service system for oil reservoir automation control.

It can be observed especially in the automation of oil and gas industry while monitoring oil fields via controlled sensors and maintaining of high quality of services (QoS) leads to improvement of the oil production rate. In other words, the stipulations of undelayed data collection and transmission pre-determine a choice of the best solutions of optimal production and forecast accuracy.

The control framework can be considered as one that finds optimal set of ICs in order to provide minimum cost and maximum performance for a desirable QoS parameters. An adaptation of the system may be performed through local interactions and, therefore, the overhead is limited by interaction with neighboring nodes. This architecture can be scaled up by allowing the deployment of multiple service instances running on different servers that are devoted for each intelligent controller.

The choice of appropriate services or the process of service selection is defined by policies that show QoS characteristics for the service such as response time, reliability, availability, and throughput. It is essential to design and implement admission control mechanism that will be able to conduct optimal service selection in order to introduce service composition framework supporting QoS. In other words, this framework should represent QoS-aware management and adaptation infrastructure that provides essential service requirements.

Therefore, we suggest that the relationship between possible service composition and QoS constraints will be incorporated into the design of a QoS-aware sensor network architecture: the additional complexity providing significant benefit at runtime through automated policy generation. Section 2 introduce MDE approach for QoS-aware system architecture. Section 3 describes automatic policy generation for the proposed architecture. Section 4 demonstrates Case study. Section 5 compares different techniques that base on policy-aware service composition. Section 6 summarizes contribution and results.

2 MODEL-DRIVEN QoS-AWARE SYSTEM ARCHITECTURE

It is quite often when control of quality of service attributes at run-time is ambitious as there are abundant calculations needed to prepare data for transmission. Moreover, the accuracy of data processing affects the future loads of the distributed system. The main goal of this paper is to apply Model-Driven QoS-aware architecture embedded into the network of controllers which is capable to provide minimum delays while transmitting and processing data and meeting desirable quality of service requirements. QoS-aware

management should provide service performance and availability monitoring as well as provision resources based on predefined policies and agreements.

This work presents a model-driven framework for automatic generation of reconfiguration policies. In particular, we suggest that the relationship between possible reconfigurations and QoS constraints should be incorporated into the design of a SOA: the additional complexity providing significant benefit at runtime through automated policy generation. Our focus will be on meeting QoS constraints (performance and reliability) for an overall architecture, what we consider as Service Level Agreements (SLAs).

Figure 2 shows our framework in large. All requests arriving to the system have to be served by all abstract services in finite amount of time according and after serving they leave the system. Every abstract service communicate with a Mediator Service that binds Actual Service from a repository. Actual Services are updating from Service Providers side. The repository is updated by Policy Service which produces optimal selection basing QoS constraints and given SLAs. In fact, Policy Service indicates to each abstract service which Actual Service to pick up.

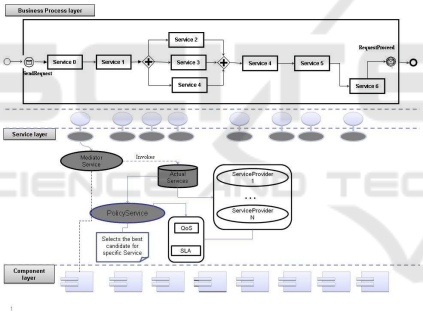

Figure 2: QoS aware service selection and policy generation mechanisms.

In our work we employ a simple generic notion of reconfiguration *policy*, by means of the metamodel extension. The metamodel identifies the architectural roles and relationships that are necessary to construct a model of the monitored system. A distinguishing feature of our metamodel is that it includes

- a QoS constraint language, based on the UML QoS profile, and
- an architectural adaptation policy definition based on dynamic programming approach from optimal control theory that serves both as a specification of how an architecture should evolve in the face of QoS constraint violations for the adaptation engine.

As it can be seen from Figure 2 the dynamic service composition can be implemented by applying policies which are based on QoS requirements and SLAs. The service composition can be considered as model transformation from design-time to run-time abstraction levels.

A policy is modeled at design-time as a possible transformation that an architecture model can undergo, representing possible reconfigurations of service composition. Therefore, service selection allows us to consider the reconfiguration of an architecture as a *transformation* from one SOA model instance to another.

The reconfiguration of an architecture realized as three Model Transformations (MT): MT1, MT2, MT3 (Figure 3).

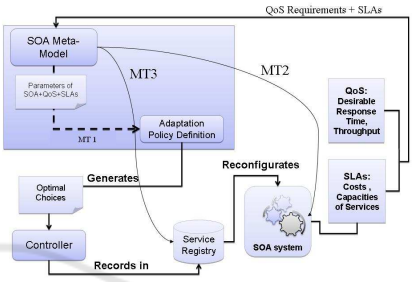

Figure 3: Model transformations for the SOA infrastructure.

The model transformation MT1 is able to automatically frame the problem as a dynamic programming optmization problem, over which our dedicated solver can determine reconfiguration strategies (choices over variant points) as a function of environmental changes. This resulting policy table is then combined with a mapping from choices to actual actions on the implemented system, to provide a runtime adaptation engine.

We employ model transformation MT2 again to extract application metadata from the design time metamodel, with the purpose of understanding how the system is configured at runtime and, consequently, what needs to be monitored:

- Deployment data for individual services (location, interfaces, etc);
- The initial architectural configuration of services (what usage connections exist between services);
- The set of queued interfaces;
- The basic properties that are necessary to compute values of the QoS characteristics used in the model.

A third model transformation MT3 is then used over this monitoring information to change the information associated with individual services in the repository model, for a roundtrip approach to regenerating policies. Policy generation, as outlined in

the next section, is expensive and re-generation need not be done frequently as service information does not change often.

3 AUTOMATIC POLICY GENERATION

3.1 QoS-aware Policy Generation

The metamodel is equipped with QoS characteristics and computes the overall cost as a function of time of architectural configurations that has to be minimized. we exhibit two examples of cost definition. As an example of the cost function of the system g can be taken sum of the cost of holding requests in queue at time step t which is formulated by the following expression:

$$g(t) = \sum_{i=0}^{n} c_i^j(t) \cdot q_i(t) \quad (1)$$

where q_i is a queue length of i-th node, c_i^j is a cost of establishing connection between nodes i and j at time Another example is when the cost of the system reflects a negotiation between cost of the system and response time. In this case a sample of the cost function can have the following form:

$$f(g(t), RT) = g(t) + W \cdot max(0, RT_{des} - RT) \quad (2)$$

where RT_{des} desirable response time, RT - response time and W - a positive number $0 \leq W \leq 1$ which is a parameter that defines to which participant of the tradeoff to give more weight. If $W \to \infty$ then second multiplier vanishes. Therefore, the system will have to be balanced between desirable response time and cost of the system.

We consider QoS constraints as a combination of one or more requirements (predicates) P_1, P_2, P_3, P_4 where P_i is one from the set: Reliability, Availability, Throughput, Response Time. We introduce Constraints() method as a rule for terms P_1, P_2, P_3, P_4:

Let
S : *System.ActualCharacteristic.QoSCharactersitic*,
C : *QoSConstraint.DesirableCharacteristic*,
$\nabla_2 = \{<, \leq, >, \geq\}$

$$P_1(S, \nabla_2, C) = S.SystemResponseTime \to \quad (3)$$
$$\to Calculate() \; \nabla_2 C.SystemResponseTime.RT$$

$$P_2(S, \nabla_2, C) = S.SystemReliability \to \quad (4)$$
$$\to Calculate() \; \nabla_2 C.SystemReliability.REL$$

$$P_3(S, \nabla_2, C) = S.SystemAvailability \to \quad (5)$$
$$\to Calculate() \; \nabla_2 C.SystemAvailability.AVL$$

$$P_4(S, \nabla_2, C) = S.SystemThroughput \to \quad (6)$$
$$\to Calculate() \; \nabla_2 C.SystemThroughput.Thrpt$$

where Calculate() function computes response time, reliability, availability and throughput.

Therefore, we can expose QoSConstraint.Constraints() as any combination L composing of following pair of predicates defined in BNF form:

$$L = P_a(S, \nabla_2, C) | L \nabla_1 L \quad (7)$$

where $\nabla_1 \in \{\vee, \wedge\}$, and $a \in \{1, 2, 3, 4\}$.

Basing on QoS requirements, an adaption of the system happens to adjust the system to the appropriate performance objective. For instance, the system adjustment objective can be formulated as a rule: "The system has to have an availability of 99.9% during the business hours of weekdays". This rule can be expressed according to (5):

$$AVL = 0.999$$

Another example of the rule to conform the system by reliability and response time requirements may sound as: "The system must be reliable no less than 95% and throughput of the system has to be no less 700 messages per second".

$$(REL \geq 0.95) \wedge (Thrpt > 700)$$

Therefore, our for given Constraints() we have to find the best set of connected nodes for each node while keeping minimum CostFunction() of the system.

In (Akzhalova and Poernomo, 2010) we proposed MDA for SOA architecture which uses automatic policy generation implemented as optimal service selection framework. We employ similar approach for the sensor network QoS-aware framework.

After cost model has chosen and QoS requirements are determined the model transformation MT1 automatically changes the system to adjust the system to desirable performance level. The system adaptation happens by calling Reconfigure() selftransformation to make the system satisfy to desirable QoS characteristics which are pre-defined in QoSConstraints. Reconfigure() generates Policy which is used then for a Binding appropriate Service to the node. In fact, Reconfigure() produces Policy as a product of the following transformation:

Reconfigure: $System \times QoSConstraints \to Policy$ (8)

where reconfiguration of System is evaluated by its cost model defined by CostFunction.

Every TimeStep when System violates QoSConstraints, Recongigure() defines Service.ID that has to be bound for each node. We designate a candidate Service as $\{Policy(TimeStep) = ID,\ ID = 1,...,Numberof Services\}$.

To find best candidate service at each time step:

$BestPolicy(TimeStep) \in \{Policy(TimeStep) = ID,$

$ID = 1,...,Numberof Services\}$

that satisfies to QoS constraints:

$$Constraints() \equiv true$$

and gives a minimum to an overall cost of the System:

$$System.CostFunction(Policy) \rightarrow \min, \quad (9)$$

where System changes its reconfiguration according to System.SystemConstraints():

$$SystemConstraints(TimeStep, Policy(TimeStep)) \quad (10)$$

The problem of optimal adaptation at time t is one of choosing the best server from the directory for each i-th service. That is, it is one of finding the best function set $u_i(t)$ that provides the lowest overall cost while meeting desirable response time. Therefore, in this case the formulation of optimal control problem will be derived from :

To find an optimal control:

$\bar{u}^*(t) \in U^*(t) = \{u_i(t) = j : j = 1,...,m, RT(\bar{x}) < RT_{des}\}$

that gives a minimum to functional:

$$J(\bar{x}, \bar{u}) \rightarrow \min, \quad (11)$$

where the configuration of the system is defined by system transformation:

$$\begin{aligned} &\bar{x}(t+1) = F(\bar{x}(t), \bar{u}(t)), \\ &\bar{x}(t) \in D, \\ &t = 0,...,T-1 \end{aligned} \quad (12)$$

The problem (sec5:eq11) - (trans1) is constrained nonlinear optimal control problem. There is no exact analytical decision of the formulated problem. In order to solve the problem it is necessary to use some of numerical optimization approaches. In the next subsection we give basics of iterative numerical methods to solve the problem and convergency definition.

There is no still agreement on the best approach of solving of the problem 12, however, traditionally, among popular methods of solving such problem are dynamic programming algorithm, genetic algorithm, simulated annealing and others. We apply dynamic programming algorithm, an iterative approach, to solve the formulated problem.

3.2 Best Candidates Selection Algorithm

Dynamic programming is a general approach to solve optimization problems that involve making a sequence of interrelated decisions in an optimum way. First, the problem is divided into subproblems (stages) where each stage has a number of states. The control (decision) at each stage updates the state into the state for the next stage. The main idea of the approach is that at given current state the optimal decision for the remaining stages is independent of decisions made in previous states.

In particular, the algorithm minimizes the sum of the cost incurred at the current stage and the least total cost that can be incurred from all subsequent stages, consequent on this decision. This principle is known as the Bellman's principle of optimality (Bellman, 1957) and dynamic programming algorithm consists of the following steps:

Dividing into Stages. The stages here related to time (hence the name is dynamic programming) and they are solving backward in time. In other words, we consider T stages: $k = T, T-1, ..., 0$.

Defining States at Stage. Each stage has a number of states each of which indicates a candidate server.

The value set of state variable x^k at stage k is the state set at stage k: $x^k = \{x(k), ..., x(T)\}$ which is a solution of the problem:

$$\begin{aligned} &x^k(t+1) = F(x^k(t), u^k(t)),\ \ t = k,...,T-1 \\ &x^k \in D^k = \{\bar{x}(t) \in D : t = k,...,T\} \end{aligned} \quad (13)$$

and correspondent control at stage k is defined as following:

$$\begin{aligned} &u^k = \bar{u}, u^k \in U^K, \\ &U^k = \{\bar{u}(t) \in U :\ \mathrm{t = k,...,T}\}. \end{aligned} \quad (14)$$

Decision at Stage. The decision at a stage updates the state at a stage into the state for the next stage. Therefore, we calculate cost at current stage k:

$$\begin{aligned} &g^k(x^k(t), \bar{u}(t)), \\ &u^k \in U^K,\ \ x^k(t) \in D^k \\ &t = k,...,T \end{aligned} \quad (15)$$

and we find the minimum of cost functional:

$$\begin{aligned} &J^k(\vec{x}^k, \vec{u}^k) = \sum_{t=k}^{T} g(\bar{x}(t), \bar{u}(t)), \\ &B^k(x^k, u) = \min_{u \in U^k} J^k(x^k, u), \\ &x^k \in D^k. \end{aligned} \quad (16)$$

where $k = T, T-1, T-2, ..., 0$.. The function B^k refers to Bellman's function.

Recursive Value Relationship. According to the fundamental dynamic programming principle of optimality given the current state, the optimal decision for the remaining stages is independent of decisions made in previous states. In other words, the optimum decision at stage k uses the previous found optima. That is recursive relationships means that a cost functional appears in both sides of the following equation:

$$B^k(x^k,u) = \min_{u \in U^k} \left\{ g^k(x^k,u) + B^{k+1}(x^k,u) \right\}$$
$$u \in U^k, \;\; k = T, T-1, ..., 0. \tag{17}$$

In the meantime, we find optimal control u^k, $k = T, ..., 0$.

Therefore, following this algorithm we find optimal policies that minimizes cost function for a given constraint.

4 EXPERIMENTS: DEPENDENCY BETWEEN SERVICE CAPACITY AND PERFORMANCE

The presented automatic policy generation approach to select best candidates for the SOA system. It was implemented as a middleware using Java JDK1.6, XML and Eclipse Modeling Tools (Eclipse Classic 3.5.2) on Intel(R)Core(TM)2Quad CPU/2.66Ghz/2.66Ghz/RAM3.23GB. A general scheme of the project can be represented in the following class diagram (Figure 4).

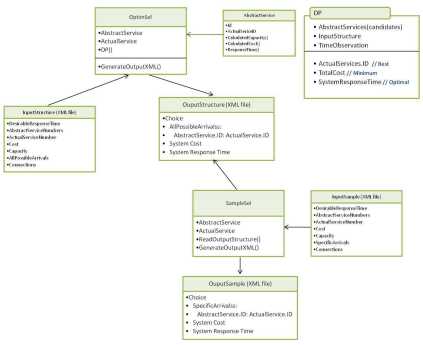

Figure 4: Class diagram of the optimal service selection implementation.

OptimSel class is responsible for generation of optimal choices for given input parameters of SOA system. Input parameters of SOA system are collected by InputStructure Class which is XML file containing QoS requirements such as Desirable Response Time, data about Actual Services number and their Capacities and Costs, Connections between services and all possible Arrivals. Those inputs are used then for a series of experiments in order to test different case studies that will be presented in the next sections. OptimSel reads (parses) InputStructure file and produce OutputStructure XML file by means of DP (dynamic programming) Method.

DP is the Method that performs dynamic progarmming algorithm using Observation Time as a number of states that was described in previous Chapter. DP returns Best Actual Service for each Abstract Service that bring to the system minimum Total Cost and Response Time of the system that is not violating Desirable Response Time.

After calling and executing DP OptimSel produces OutputStructure by calling GenerateOutputStructure() method. GenerateOutputStructure() creates XML file by parsing results to specific attributes. In fact, OutputStructure file contains optimal choices for each Service per each Arrival in order to proceed efficiently arriving to the system requests.

The diagram in Figure 4 contains SampleSel, a derived class from OptimSel, that is used for each case study. For instance, if we simulate the SOA system for different distribution of arrivals we use policies generated and stored in OutputStructure XML and generate table specifically for those arrivals. The table for those arrivals is OutputSampleXML file.

In order to test the system for varying values of Desirable Response time, Capacities, Costs or Connections it has to be generated relative OutputStructure XML files for each case.

The purpose of this experiment is to find out how capacities of the service affect to the cost and performance of the system.

The system is composed of $n = 8$ semantic services and each service has $m = 6$ independent services. Assume that the service at each Semantic service has cost defined by Cost c_i^j and Capacity $\tilde{\mu}_i^j$ (i = 1, .., 8; j = 1, .., 6). The values of Cost and Capacities are shown in Table 1 and Table 2, respectively.

In order to conduct the test we build new tables of capacities which range over average absolute deviation *MD* from previous capacities. An average absolute deviation is also often called Mean absolute Deviation (MD) is the mean of the absolute deviations of a set of data about the data's mean. The *MD* of the set data size n is defined by:

Table 1: Maximum capacity of Services at each Semantic Service (requests per second).

ID of the Service	ID of the Semantic Service							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	50	55	55	57	65	57	65	67
2	51	60	65	58	67	60	67	68
3	56	67	71	64	70	63	72	70
4	60	70	72	68	72	65	74	71
5	80	85	90	70	80	68	75	75
6	85	90	120	100	120	100	85	84

Table 2: The cost of the Service for each Semantic Service (unit of money per request).

ID of the Service	ID of the Semantic Service							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	10	11	15	10	15	16	12	14
2	20	21	23	20	23	25	18	15
3	30	31	35	28	35	30	23	20
4	40	42	45	35	42	40	35	30
5	50	52	55	43	44	41	40	40
6	52	55	60	45	55	45	51	50

$$MD = \frac{1}{n}\sum_{i=1}^{n}\{\alpha_i - \overline{\alpha}\}$$

where $\overline{\alpha}$ - the mean of the distribution:

$$\overline{\alpha} = \frac{1}{n}\sum_{i=1}^{n}\alpha_i$$

Basing on investigation of relationship between *MD* variation and performance of the system we may take this dependency as the next step to study cost functional of the system that can adjust model validation. We consider a system that process Poisson distributed requests (Figure 5) that has to adapt itself to meet $RT < RT_{des}$ constraint where RT_{des}=4 seconds.

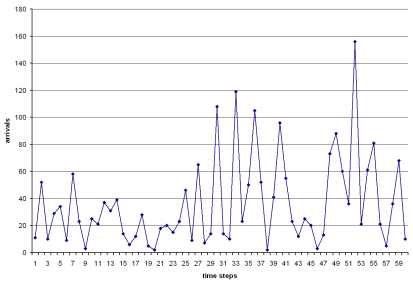

Figure 5: Arrivals to the system.

In fact, handling different capacities of the service, we obtain three systems with services that have different Capacities and Costs. In other words, one can represent those variations as different offers from a number of service providers. Therefore, we examine how different offers may affect to the overall responsiveness and expenses of exploiting systems. In particular, capacities of systems where $MD = -11$, $MD = +11$ are defined in Table 3 - Table 4. We leave the cost of the services fixed (see Table 2).

According to given data we apply dynamic programming algorithm that finds optimal policies for each system. Analyzing how capacities will affect to response time of the system and total cost on Figure 6 and Figure7 we may discover that in general evolution of the systems with different capacities have similar dynamics but different amplitude. For instance, response time obtained by dynamic programming algorithm with higher capacities $+MD$ has the smallest values in comparison with one that was produced by applying services with lower capacities.

Instead the policy that had to select among services with capacities $-MD$ from original transforms the system to the state when one has slowest responsiveness. In other words, the new configuration has violated desirable response time. Presumably, existing resources were not sufficient to tackle with 156 requests at time step 52. At the same time, this outcome shows that in order to apply dynamic programming there is needs to put a constraint on capacities distribution in advance to avoid non-optimal solutions.

Surprisingly, the maximum response time returned by the policy with originally capacities is very close to the value of response time of the system with capacities $+MD$. For instance, in case of highest workload at time step 52 when 156 requests had to be proceed. In other words, having same cost of the service and capacities that differ on 11% from *MD*, we observe that dynamic programming produces re-

Table 3: Maximum capacity $+MD$ of Services at each Semantic Service (requests per second).

ID of the Service	ID of the Semantic Service							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	61	66	66	68	76	68	76	78
2	62	71	76	69	78	71	78	79
3	67	78	82	75	81	74	83	81
4	71	81	83	79	83	76	85	82
5	91	96	101	81	91	79	86	86
6	96	101	131	111	131	111	96	95

sponse time as close as possible to the desirable one. Therefore, the constraint formulated as inequality for dynamic programming is the objective to reach as near as possible. However, the picture of costs of cases with original and capacities $+MD$ is dissimilar.

In particular, as it can be observed in Figure7 the cost of the system in case of exploiting services with higher capacities, was the lowest one. In contrast to the similarity of response times of the system with original capacities and capacities $+MD$, the cost of the system with original capacities at time step 52 is the highest one. In the meantime, when system handles services with lowest capacities $-MD$ the cost of the system was expensive as it had to afford a range of workloads by using less efficient resources.

Table 5 displays maximum achieved response time for different systems and their total cost. As it can be studied from the Table and examining all accommodated systems we have detected that the best variant in terms of minimum cost and response time was generated by policy that operates with services with highest capacities.

As it was mentioned at the beginning of this test, Table 2 remains having constant values of service cost for all considered systems. We may make preliminary assumption that total cost of the system decreases while handling services when the value of their capacities increases. Therefore, there is an inverse relationship between performance of existing resources (services) and cost and response time of the system.

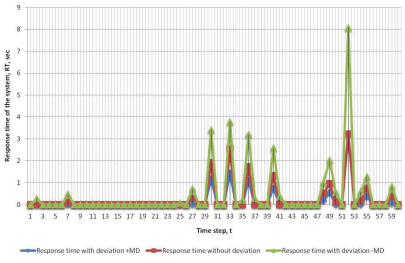

Figure 6: Dynamics of response time for different sets of capacities.

Summarizing the case study, we have investigated

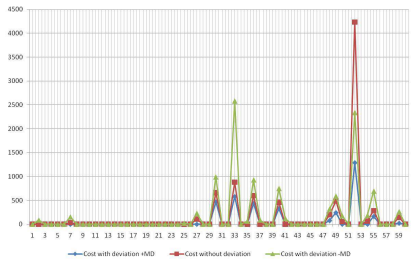

Figure 7: Dynamics of cost of the system for different sets of capacities.

an influence of changing capacities to the system characteristics:

- The experiments showed a considerable impact of changing capacities to the cost of the system. and response time at given constraints.
- If the deviation MD is bigger than 20% then the response time of the system differs significantly than one produced handling original service capacities.
- Employing dynamic programming to the systems with capacities that have $MD + 11\%$, the algorithm gives maximum response time that is very close to one obtained by applying original capacities. In fact, in both cases it attempts to approach to the desirable response time.
- On other hand, if value of capacities less than enough then constraint might be violated.

These outcomes can deliver essential recommendations when system's resources struggle with workloads and their utilization changes from current level. At the same time, these results can used by service providers to install required service contract with service consumers.

5 DISCUSSION

Lymberopoulos Leonidas, Emil Lupu and Morris Sloman in (Lymberopoulos et al., 2003) propose a framework that supports automated policy deployment and

Table 4: Maximum capacity $-MD$ of Services at each Semantic Service (requests per second).

ID of the Service	ID of the Semantic Service							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	39	44	44	46	54	46	54	56
2	40	49	54	47	56	49	56	57
3	45	56	60	53	59	52	61	59
4	49	59	61	57	61	54	63	60
5	69	74	79	59	69	57	64	64
6	74	79	109	89	109	89	74	73

Table 5: The maximum response time and cost of the system for different types of policies.

Type of system	**Maximum response time, Max(RT)**	**Total cost of the system**
Services with capacities $-MD$	8.069	10591
Services with original capacities	3.232	8209
Services with capacities $+MD$	3.225	3616
Services with capacities $+2*MD$	2.1099	2494
Services with capacities $+3*MD$	1.4314	1607

flexible event triggers to permit dynamic policy configuration. Basically, the authors developed PONDER policy language for adaptation at the service layer to select and modify policies at the network layer (Damianou et al., 2001). The language provides reuse by supporting definition of policy types, which can be instantiated for each specific environment. The proposed adaptation is run-time and it also allows to build a new adaptation strategies that can be incorporated into the management system by adding new policies which react to different events using the existing policy actions or by replacing existing policies with new versions, which either implement new actions on the managed objects or new actions on the Policy Management Agents. In general, an adaptation is provided in one of the following ways:

- by dynamically changing the parameters of a QoS policy to specify new attribute values for the run-time configuration of managed objects;
- by selecting and enabling/disabling a policy from a set of pre-defined QoS policies at run-time.

The advantage of the presented approach is that parameters of the selected network QoS policy are calculated and set at run-time. The authors addressed the future research on developing techniques that provide policy specification and adaptation across different abstraction layers; and to develop tools and services for the engineering of policy-driven systems.

The research of QoS policy based Web Service selection conducted by (Wang et al., 2006) attempts to generalize QoS contract specification, establishment, and monitoring for Service Level Management into one large framework. The authors introduce a QoS management architecture consisting of component services, their interactions, and interfaces with external services such as real-time host and network condition monitoring (through COTS Monitoring tools like Empirix OneSight/FarSignt) (Wang et al., 2005). The framework includes a tool for end users to generate, modify and validate QoS specifications in the given language. The tool facilitates generation of the QoS specifications without requiring the user to remember the supported set of QoS characteristics and their value domains. The Monitoring Service registers condition predicates with the Diagnostic Service, which returns with notifications after that Monitoring Service updates the corresponding data in Maintenance Service, which in turn activates some Adaptation Mechanisms defined in the policy.

Therefore, the framework provides reusable concepts and processes to facilitate QoS contract establishment and monitoring through contract negotia-

tion, resource management, diagnostics and adaptation mechanisms. However, as the authors note in (Wang et al., 2006), it has to be done the further research and development of dynamic QoS-driven resource management algorithms for Service Level Management.

This work presents a way to tackle with above problems by introducing model-driven approach to SOA together with the optimal control technique as transformation for the SOA meta-model in order to automatically reconfigure the system in less costly way.

6 CONCLUSIONS

The policy generation was formulated as an optimal control problem which allows automatically generate appropriate configuration of the service-oriented system to meet QoS constraints. It means that proposed approach is generic in terms of choosing different level of QoS requirements and applying different techniques to solve the optimal control problem. The model is extensible as the QoS requirements may be included during system design. We have offered dynamic programming approach as the solution of the formulated problems. The case study investigates how different parameters of the service-oriented system and constraints affect to the performance and dynamics of system utilization. We have evaluated obtained results and formulated recommendations and best strategies for employing dynamic programming approach to dynamically adapt SOA according to desirable QoS characteristics.

REFERENCES

Akzhalova, A. and Poernomo, I. (2010). Model driven approach for dynamic service composition based on qos constraints. *Services, IEEE Congress on*, 0:590–597.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Damianou, N., Dulay, N., Lupu, E., and Sloman, M. (2001). The ponder policy specification language. In Sloman, M., Lobo, J., and Lupu, E., editors, *POLICY*, volume 1995 of *Lecture Notes in Computer Science*, pages 18–38. Springer.

Lymberopoulos, L., Lupu, E., and Sloman, M. (2003). An adaptive policy-based framework for network services management. *J. Netw. Syst. Manage.*, 11(3):277–303.

Martin Keen, Amit Acharya, e. a. (2004). *Patterns: Implementing an SOA Using an Enterprise Service Bus*. IBM Corp., Riverton, NJ, USA.

Wang, C., Wang, G., Wang, H., Chen, A., and Santiago, R. (2006). Quality of service (qos) contract specification, establishment, and monitoring for service level management. In *EDOCW '06: Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, page 49. IEEE Computer Society.

Wang, G., Wang, C., Chen, A., Wang, H., Fung, C., Uczekaj, S., Chen, Y.-L., Guthmiller, W. G., and Lee, J. (2005). Service level management using qos monitoring, diagnostics, and adaptation for networked enterprise systems. In *EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, pages 239–250. IEEE Computer Society.