UNIVERSITÀ DEGLI STUDI DI BERGAMO
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
E METODI MATEMATICI°

QUADERNI DEL DIPARTIMENTO

Department of Information Technology and Mathematical Methods

Working Paper

Series "*Information Technology*"

n. 2/IT – 2011

A framework for adapting service-oriented applications based on functional/extra-functional requirements tradeoffs: the Stock Trading System case study

by

R. Mirandola, P. Potena, P. Scandurra, E. Riccobene

° Viale Marconi. 5, I – 24044 Dalmine (BG), ITALY, Tel. +39-035-2052339; Fax. +39-035-562779

[§] L'accesso alle *Series* è approvato dal Comitato di Redazione. I *Working Papers* della Collana dei Quaderni del Dipartimento di Ingegneria dell'Informazione e Metodi Matematici costituiscono un servizio atto a fornire la tempestiva divulgazione dei risultati dell'attività di ricerca, siano essi in forma provvisoria o definitiva.

A framework for adapting service-oriented applications based on functional/extra-functional requirements tradeoffs: the Stock Trading System case study

Raffaela Mirandola
Politecnico di Milano
DEI, Milano, Italy
mirandola@elet.polimi.it

Pasqualina Potena, Patrizia Scandurra
Università degli Studi di Bergamo
DIIMM, Dalmine (BG), Italy
{*pasqualina.potena,patrizia.scandurra*}*@unibg.it*

Elvinia Riccobene
Università degli Studi di Milano
DTI, Crema (CR), Italy
elvinia.riccobene@unimi.it

***Abstract*—This report presents the model parameters and experimental results for adapting a sample service-oriented application from a Stock Trading System. In particular, the followed adaptation strategy combines the metaheuristic *steepest-ascent hill-climbing* with some *tactics* (i.e. extra-functional adaptation patterns) to increase the system availability and performance.**

I. The STS case study

We illustrate the adaptation (both in structure and in behavior) of a sample application from the Stock Trading System (STS) in [3] by exploiting the metaheuristic *steepest-ascent hill-climbing* [7] and some *tactics* as concrete examples of extra-functional adaptation patterns. Specifically, first we describe the application of the metaheuristic search technique, and then we show the use of some tactics to increase the system availability and performance.

Fig. 1 shows the architecture of the considered application of the STS in terms of an SCA [5] assembly. Briefly, an STS user, through the `OrderWebComponent` interacting with the `OrderDeliveryComponent`, can check the current price of stocks, placing buy or sell orders and reviewing traded stock volume. Moreover, he/she can know stock quote information through the `StockQuoteComponent`. STS interacts also with the external Stock Exchange system, which we do not model.

Fig. 2 shows a fragment of the (abstract) specification for the `OrderDeliveryComponent` behavior using the *Abstract State Machines* (ASMs)[1] formal method [10], which is able to model service interactions, orchestrations, compensations, and the services internal behavior [1], [6]. The main service of this component (the rule `r_place` annotated with `@service`) is to place buy or sell orders when requested (see the blocking receive action and the replay action preceding and following, respectively, the service invocation within the component's main rule `r_OrderDeliveryComponent`). The ASM definition for the provided and required interfaces of the `OrderDeliveryComponent` are reported in Fig. 3. They are ASM modules containing only declarations of business agent types (the subdomains `OrderDelivery` and `StockExchange` of the predefined ASM `Agent` domain) and of business functions (parameterized ASM out functions) used as temporary locations to store service computation results.

[1]ASMs are an extension of FSMs: *states* are arbitrary complex data (multi-sorted first-order structures) and the *transition relation* is specified by *rules* describing how functions change from one state to the next.

A. Model parameters of the STS

This section reports the values of the parameters for the STS example. In order to derive these system parameters, we took inspired from the WEB-based data retrieval system adopted as case study in [2]. In such a system clients are equipped with the local processing capability and local database. A WEB connection is provided to execute data search and data updates in a distributed system, through the network. We have assumed, though not shown in Figure 1, that our STS application also instantiates an agent on the server side of the WEB-based data retrieval system.

As done in [2], the estimation of the parameters entering our system has been partly based on the monitoring of an existing data retrieval system at University of L'Aquila, partly extracted from software artifacts of the same system.

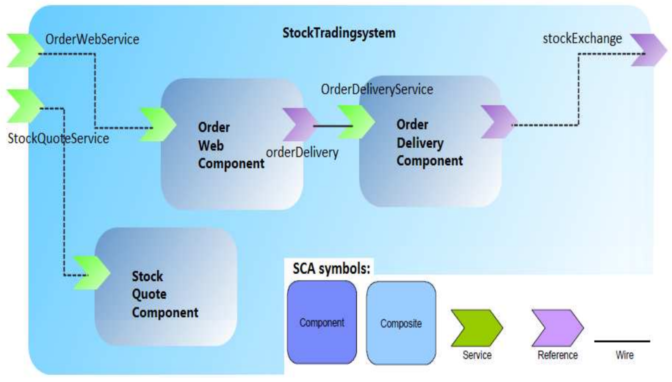

Figure 1. Stock Trading System

	Instance alternatives	Cost c_{ij}	Average delivery time d_{ij}	Average no. of invocations s_i	Prob. of fail. on demand μ_{ij}
C_1	C_{11}	1	4	180	0.0002
	C_{12}	2.5	4	180	0.00001
	C_{13}	2	4	180	0.0004
C_2	C_{21}	2	4	20	0.0002
	C_{22}	3	4	20	0.00005
	C_{23}	6	6	20	0.00001
C_3	C_{31}	10	4	60	0.0003
	C_{32}	14	10	60	0.0007
	C_{33}	10	10	60	0.00006

Table I
EXAMPLE OF PARAMETERS OF INSTANCE AVAILABLE FOR EXISTING COMPONENTS

```
module OrderDeliveryComponent
//@Provided service interface
import OrderDeliveryService
//@Required service interface
import StockExchangeService
...//Other module imports
signature: //ASM function declarations
//@Reference to the external stock exchange system
shared stockExchange:Agent−>StockExchange
//@Backref back reference to the requester
shared client: Agent −> Agent
//Other function declarations for internal computation
controlled order: Agent −> Order
definitions:
//ASM rules for the provided service operations
@Service
rule r_place($client in Agent,$o in Order)= ... //to place buy or sell orders
...
//ASM rule for the component’s agent behavior
rule r_OrderDeliveryComponent=
 seq
   r_wreceive(client(self),”place”,order(self))
   //direct service invocation
   r_place(client(self),order(self))
   r_wreplay(client(self),”place”,order(self))
 endseq
//constructor rule
rule r_init($agent in OrderDelivery)=$ ... //do initialization (if any)
```

Figure 2. ASM module of the `OrderDeliveryComponent`

```
module OrderDeliveryService
import ... //Other module imports
signature:
// the domain defines the type of the provider component’s agent
domain OrderDelivery subsetof Agent
// business function value
out place: Prod(Agent,Order) −> Order

//@Remotable
module StockExchangeService
import ... //Other module imports
signature:
domain StockExchange subsetof Agent
out sendOrder: Prod(Agent,Order) −> Rule
```

Figure 3. ASM modules of the `OrderDeliveryComponent` interfaces

However, incomplete documentation forced to adopt extrapolation techniques for providing certain values. For example, the number of invocations has been obtained by analyzing partial scenario descriptions and validating the analysis results with monitored average number of interactions.

Table I-B shows an example of parameters value (which we have used for the *steepest-ascent hill-climbing* application presented in Section I-B) for instance available for the `OrderWebComponent` (i.e., C_1), `StockQuoteComponent` (i.e., C_2) and `OrderDeliveryComponent` (i.e., C_3).

The second column of Table I-B lists, for each component, the set of instance alternatives available. For each alternative: the adaptation cost c_{ij} (in KiloEuros, KE) is given in the third column, the average delivery time d_{ij} (in days) is given in the fourth column, the average number of invocations of the component in the system si is given in the fifth column, finally the probability of failure on demand μ_{ij} is given in the sixth column.

Although the estimate of c_{ij} is outside the scope of our work, we remark that it may be estimated in different ways. It could be estimated, for example, as a function of the price charged for each invocation its services, the cost for integrating the component instance into the system (e.g., the cost needed to handle mismatches between the functionalities offered by the alternative and the functional requirements required for it). The delivery time d_{ij} of component might be decomposed in the sum of the time needed to the vendor to deliver the component, and the adaptation time. The parameter s_i represents the average number of invocations of a component within the system execution scenarios. This value is obtained as a function of the execution states of the component (i.e., using the ASM rules regarding service invocation, components interaction, error and compensation handling). The number of invocations is averaged overall the scenarios by using the probability of each scenario to be executed. The latter is part of the operational profile of the application. Finally, μ_{ij} represents the probability for the alternative j of the component i to fail in one execution (see, [2] for more details).

Note that we have defined the parameters of components as average values of the values of their provided services. The parameters could be refined with respect to the services without essentially changing the overall model structure.

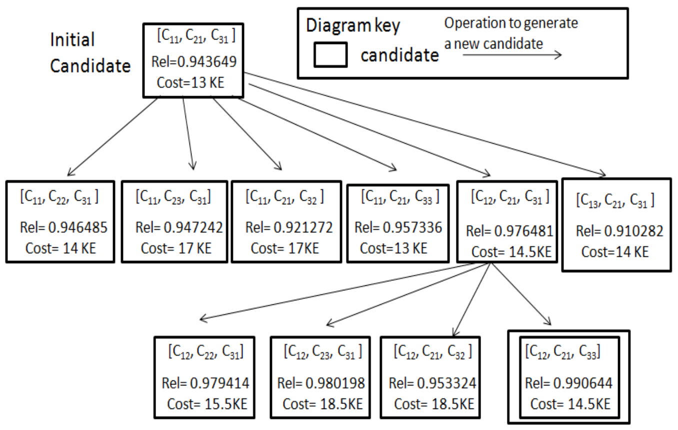
Initial Candidate
[C11, C21, C31] Rel=0.943649 Cost=13 KE
Diagram key
candidate
Operation to generate a new candidate
[C11, C22, C31] Rel= 0.946485 Cost= 14 KE
[C11, C23, C31] Rel= 0.947242 Cost= 17 KE
[C11, C21, C32] Rel= 0.921272 Cost= 17KE
[C11, C21, C33] Rel= 0.957336 Cost=13 KE
[C12, C21, C31] Rel= 0.976481 Cost= 14.5KE
[C13, C21, C31] Rel= 0.910282 Cost=14 KE
[C12, C22, C31] Rel= 0.979414 Cost= 15.5KE
[C12, C23, C31] Rel= 0.980198 Cost= 18.5KE
[C12, C21, C32] Rel= 0.953324 Cost= 18.5KE
[C12, C21, C33] Rel= 0.990644 Cost= 14.5KE

Figure 4. Example of steepest-ascent hill-climbing application

B. Methaeuristic search

Several metaheuristics [7] with different characteristics could be adopted depending on the problem; for example, heuristics considering that if the reliability of the most used components increases then the reliability of the whole system will likely increase. As remarked in [4], there exist design options for which we have no prior knowledge on how they affect the extra-functional property for a particular system. To this extent, undirected operations could be performed (e.g., random choices or exhaustive evaluation of all neighboring candidates).

Figure 4 shows an example of instantiation of our optimization approach by considering, on the STS example, the *steepest-ascent hill-climbing* metaheuristic [7] that tries to adapt the system minimizing the adaptation costs and assuring a level of system reliability greater than 0.98. The initial candidate is the vector $[C_{11}, C_{21}, C_{31}]$ (see Figure 4), where C_{ij} denotes the jth instance available on the market for the component C_i with C_1 indicating the `OrderWebComponent`, C_2 the `StockQuoteComponent` and C_3 the `OrderDeliveryComponent`. Each vector came with two parameters: the resulting system reliability and the cost of the solution (predicted using the reliability and cost model used in [2] and reported in Table). At each iteration step, a set of new candidates is generated by replacing, one per time, an existing component with one available on the market. The best candidate is then selected as the one improving the system reliability and minimizing the adaptation costs. It becomes the basis for next candidates generation. The process terminates either if no better candidates can be found or the reliability threshold is reached. In our case, the optimization process returns the solution $[C_{12}, C_{21}, C_{33}]$ with reliability equals to 0.990644 and cost equals to 14.5 KE.

We have also implemented the optimization process with the multi-objective optimization [8]. We have formulated the function objective as weighted sum, $\sum_{i=1}^{n} \alpha_i \cdot f(i)$ where α_i's are real numbers of the set of objectives (i.e., $f(i)$), to minimize (e.g., adaptation cost, probability of failure and response time), and it holds: $\sum_{i=1}^{n} \alpha_i = 1$. To determine the set of solutions (i.e., an approximation of the Optimal Pareto set) with respect to some values of α_i's, after fixed the values of the weights α_i and defined the objective function as the weighted sum of the functions that must be minimized, we have applied a metaheuristic technique.

For example, if the process with the *steepest-ascent hill-climbing* (see Figure 4) has the aim to minimize the adaptation cost and the delivery time of the system with weights α_i fixed to 0.2 and 0.8, respectively (i.e., more importance is assigned to the delivery time attribute), it would select the solution $[C_{12}, C_{21}, C_{33}]$ rather then $[C_{12}, C_{23}, C_{31}]$.

Moreover, depending on several factors due mainly to the use of our framework for evolution (at re-design time) or self-adaptation (at run time), the optimization process could adopt different analysis strategies. For example, for assuring a certain reliability and response time to a safe-critical system the re-deployment of the system components (i.e., SCA domain actions) should be driven by a reliability model embedding also the probability of failure of communication wires (i.e., SCA bindings[2]) connecting the components. As an example, let us consider the reliability of the system by taking into account the probability of failure of the wire connecting the `OrderWebComponent` and the `OrderDeliveryComponent`. In this case, if, for example, they are deployed in different hosts and their average number of interactions is equals to 104 and they are connect through a link with a probability of failure equals to 0.0001, the reliability of the system with respect to the solution $[C_{12}, C_{21}, C_{33}]$ would decrease from 0.990644 to 0.9803947.

C. Application of extra-functional adaptation patterns

New candidates may be generated by applying extra-functional patterns, i.e. tactics[3]. Several tactics with different characteristics could be adopted depending on the desired quality. For example, the *Resource Arbitration tactic* [3] is often used to improve performance by scheduling requests for expensive resources (e.g., processors, networks).

We here focus on the use of some tactics for addressing availability and performance requirements of the STS example. Let us assume the following extra-functional requirements (taken from [3]):

NFR1. *The STS should be available during the trading time (7:30 AM6:00 PM) from Monday through Friday. If there is no response from the system for 30 s, the STS should notify the administrator.*

[2]Services use bindings to describe the access mechanism (e.g., WSDL, JMS, etc.) that clients use to call the service.

[3]Tactics are reusable architectural solutions for issues pertaining to quality requirements such as reliability, performance, etc.

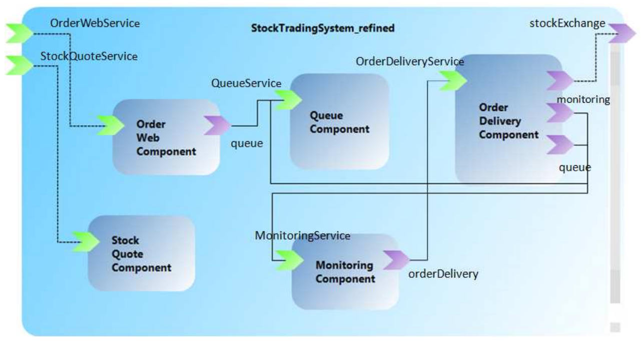

Figure 5. Adapting the STS by applying tactics for NFR1 and NFR2

```
module OrderDeliveryComponent
...
//ASM rule for the OrderDeliveryComponent's agent behavior
rule r_OrderDeliveryComponent=
... seq
        par //Queue consuming
          r_wsendreceive[client(self),"dispatch","Stock",stockorder(self)]
          r_wsendreceive[client(self),"dispatch","Option",optionorder(self)]
          r_wsendreceive[client(self),"dispatch","Future",futureorder(self)]
        endpar
        par //Order sending to the Stock Exchange system
          r_wsend(stockExchange(self),"sendOrder",(self,stockorder(self)))
          r_wsend(stockExchange(self),"sendOrder",(self,optionorder(self)))
          r_wsend(stockExchange(self),"sendOrder",(self,futureorder(self)))
       endpar
     endseq ...
```

Figure 6. The refined behavior of the `OrderDeliveryComponent`

NFR2. *The system should be able to process 300 transactions per second, 400,000 transactions per day. A client may place multiple orders of different kinds (e.g., stocks, options, futures), and the orders should be sent to the system within 1 s in the order they were placed.*

To address NFR1 the *Fault Detection Tactic* for the detection and notification of a fault to a monitoring component or to the system administrator can be adopted. Such kind of tactic can be refined into other ones (e.g., *Ping/Echo*, *Heartbeat* and *Exception* tactics [3]). As done in [3], we support NFR1 combining *Ping/Echo* and *Heartbeat*.

As in [3], NFR2 is supported combining the *FIFO* (for the Resource Arbitration) and *Introduce Concurrency* (for the Resource Management) tactics. The *FIFO* tactic allows clients to place each type of orders (e.g., stocks, options, futures) to a dedicated queue for immediate processing. Finally, to handle considerable amount of transactions by their kinds within a very short time, as suggested in [3], NFR2 can be also supported by reducing the blocking time of transactions on I/O, which can be realized by the combined use of the *FIFO* and *Introduce Concurrency* tactics (i.e., by concurrent dispatching of the same kind of orders).

Fig. 5 shows how it would change the SCA assembly by composing these tactics: the assembly is extended to add the new `Queue` component (for the *FIFO* tactic) and the `Monitor` component (for the *Fault Detection Tactic*). The `OrderWebComponent` is refined for concurrently producing orders to place into the `Queue`. Similarly, the `OrderDeliveryComponent` is refined for adding the monitoring functionality and for the concurrent consuming of different kinds of orders placed into the `Queue` component. Of course, this implies a change of the components shape (i.e. in the required/provided interfaces) and of their behavior. The behavior, for example, of the `OrderDeliveryComponent` is refined in ASM as shown in the fragment reported in Fig. 6. Essentially, the consuming and sending of different kind of orders (stock, option, or future) are executed in parallel (i.e. concurrently) by the `par` rule.

It is possible to prove that the behavior of the `OrderDeliveryComponent` in Figure 6 is a correct refinement [10] of that in Figure 2, and, therefore, all initial functional requirements are still guaranteed.

In our approach, other than composing design solutions, we perform a functional and extra-functional analysis on the resulting architecture candidates. For example, in the STS example it could be necessary for the extra-functional requirement NFR2 to check the throughput of the system to guarantee that the system processes, 300 transactions per second and 400,000 transactions per day. To this purpose, different performance analysis strategies could be leveraged: for example a transformation into a discrete-event simulation or LQNs to derive response times, throughputs, and resource utilizations as proposed in [11]. This further extensive analysis is not the focus of this report.

The impact of the adoption of the tactics should be quantified with respect to the existing system quality. For example, the introduction of new components could decrease the maximum level of reliability. In the STS example, after the embedding of new components into the `OrderDeliveryComponent` for NFR1, if the probability of failure of the instance available for this component increases (for example, from 0.00006 to 0.0002 Table I-B), then the reliability of the overall solution will decrease (from 0.990644 to 0.970639 Table I-B). Therefore, it could happen that the reliability constraint is not satisfied any more (in the example indeed, the system reliability is not greater than 0.98). Note that, also the reliability of the new `Queue` component may contribute to decrease the system reliability.

D. Lessons Learned

The importance of existing design solutions By observing our experimentations we have noticed that the adoption of existing design solutions produces results which exceed the one produced by applying only simple modifications, which are typically required for the system self-adaption.

Referring to our example, the availability of the candidate solution $[C_{12}, C_{21}, C_{33}]$ (see Figure 4) with respect to the functionality of the system "*Requiring current stock price* *and stock quote information*" is equals to 0.937587 by assuming that the availability of first, second and third component is equal, respectively, to 0.98, 0.97 and 0.94. If instead of replacing the `OrderWebComponent` with its third available instance (i.e., C_{33}) the *Recovery Preparation* and *Repair* tactics (concerned with recovering and repairing a component from a failure [3]) are applied, the system availability would increase from 0.937587 to 0.955428. In fact, the availability of `OrderWebComponent` depends on of the number of its redundant instances and it becomes 0.97. However, in this case, increasing the availability may lead to an increment of the adaptation costs. Obviously, the efficiency of design solutions application may be improved with the use of more accurate functional and non-functional analysis strategies.

Why our framework supports the works of maintainers Our framework is a valid support for the maintainers because it adapts a system without taking practically no noticeable time on standard computing equipment. The application of the steepest-ascent hill climbing approach to our example required only few minutes. The optimization process allows the automatic exploration of the search space by taking at the same time into account both the available adaptation actions and the features of the system. Obviously, sometimes the direct interaction between the framework and the maintainer could be required. For example, the application of an anti-pattern solution may require manual interaction because, a performance model of the system usually does not contain enough information to decide whether it is applicable [11]. However, such a kind of manual interaction may be required only for system evolution where speedy answers are not essential, whereas it is not required for self-adaptation.

Using our automated framework, the work of a maintainer can be facilitated. In fact, to adapt a system it is not necessary to insert as input value architectures satisfying all the required changes, as it is typically done in the state-of-the art methods, but a maintainer may suggest possible adaptation actions for each changes and the framework itself generates the necessary adaptation actions (e.g., using service selection, service re-deployment, tactics and design patterns).

For complex systems the search space is really large and complex, with the consequence that for the maintainers the space is really unmanageable and it is hard to make the right decision without some form of automated support.

References

[1] E. Riccobene, P. Scandurra, F. Albani, "A modeling and executable language for designing and prototyping service-oriented applications," in Proc. SEAA 2011.

[2] V. Cortellessa, F. Marinelli, and P. Potena, "An optimization framework for "build-or-buy" decisions in software architecture," *Computers & OR*, vol. 35, no. 10, pp. 3090–3106, 2008.

[3] S. Kim, D. Kim, L. Lu, and S. Park, "Quality-driven architecture development using architectural tactics," *Journal of Systems and Software*, no. 8, pp. 1211–1231, 2009.

[4] H. K. A. Martens, "Automatic, model-based software performance improvement for component-based software designs," in *Proc. of FESCA 2009*, vol. 253, no. 1, 2009, pp. 77 – 93.

[5] "Service Component Architecture (SCA) `www.osoa.org`."

[6] E. Riccobene and P. Scandurra, "Specifying formal executable behavioral models for structural models of service-oriented components," in *Proc. ACT4SOC 2010*.

[7] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, 2003.

[8] Y. Censor, "Pareto optimality in multiobjective problems," *Applied Mathematics & Optimization*, vol. 4, pp. 41–59, 1977.

[9] O. Zimmermann, J. Grundler, S. Tai, and F. Leymann, "Architectural decisions and patterns for transactional workflows in soa," in *ICSOC*, 2007, pp. 81–93.

[10] E. Börger and R. Stärk, *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003.

[11] A. Martens and H. Koziolek, "Performance-oriented Design Space Exploration," in *Proc. of WCOP'08*, 2008.