An Extensible Framework for Dynamic Market-Based Service Selection and Business Process Execution

Ante Vilenica, Kristof Hamann, Winfried Lamersdorf, Jan Sudeikat, Wolfgang Renz

An Extensible Framework for Dynamic Market-Based Service Selection and Business Process Execution

Ante Vilenica[1], Kristof Hamann[1], Winfried Lamersdorf[1], Jan Sudeikat[2], and Wolfgang Renz[2]

[1] Distributed Systems and Information Systems
Department of Informatics, University of Hamburg
http://vsis-www.informatik.uni-hamburg.de/

[2] Multimedia Systems Laboratory
Hamburg University of Applied Sciences
http://cms-server.ti-mmlab.haw-hamburg.de/

Abstract. Business applications in open and dynamic service markets offer great opportunities for both consumers as well as for providers of services. However, the *management* of related business processes in such environments requires considerable (often still manual) effort. Specific additional challenges arise in highly dynamic environments which may lead, e.g., to service failures or even to complete disappearance of partners and, consequently, a need to reconfigure related processes. This work aims at generic software support for addressing such challenges mainly by providing an extensible negotiation framework which is capable of performing the tasks of service selection and service execution automatically. Its technical basis are augmented, reusable and highly autonomous service components that can be tailored towards the specific needs of each business process. In addition, the implementation of the negotiation framework includes a simulation component which offers convenient means to study the outcome of different settings of the business environment a priori.

Keywords: Service-Oriented Computing, Autonomous Components, Negotiation Framework

1 Introduction

State-of-the-art business information systems rely on a high number of different services from various sources as well as on open and flexible business procedures making use of them. Software support for the development of such (distributed) applications profits substantially from a service-oriented software architecture which provides appropriate architectural patterns for the sophisticated and flexible development of information systems that can easily interoperate, change components and therefore cope with the high complexity, dynamicity and demands of modern business environments. In such scenarios, services may typically appear and disappear at any time and may, furthermore, dynamically

change their configuration, e.g. their commitment towards a certain application or their respective costs. Such kind of behaviour can be found in particular in domains that use market-based pricing for negotiating contracts between consumers and providers of respective services. These business domains require open market systems with standardized interfaces and various vendors in order to work properly and to have a competitive market. At the same time, the inherent dynamics of such market-based systems complicate the runtime management of applications composed of different services and, thus, require solutions capable to cope with these challenges as *autonomously* as possible.

For example, think of a computer manufacturer who regularly needs to acquire different well-defined and standardized sub-components like hard-disks, keyboards, RAM etc. for his production line. This scenario reveals a clear demand for an (autonomous) management of this supply process that takes into account different functional and non-functional aspects and that (dynamically) selects the most appropriate service provider at any time in a constantly changing environment. Furthermore, not only the selection but also the execution of the selected service should be automatically monitored and managed – such that, for instance in the case of a service-breakdown, appropriate actions, e.g. instantiation of a new selection process, should be initiated automatically.

Aiming at such scenarios, the work reported in this paper addresses these challenges by proposing a supportive software framework capable of facilitating the autonomous and dynamic selection and execution of service-based applications. It leverages different market-based negotiation protocols in order to determine contract partners as well as various utility-functions for specifying preferences among (potentially conflicting) objectives. Therefore, the approach proposed here enables existing as well as new composed services with the capability to participate in such automatic and autonomous service selection and process execution. Technical basis for the implementation of such dynamic and adaptive service composition is an environment that integrates workflow execution with autonomous software agents [**?**]. In this approach, software agents represent services which participate in dynamically adaptive business processes. The necessary adaptivity is realised by an underlying management middleware that controls the coordination of all participating software agents. This, in turn, leads to adaptive and autonomous properties of the workflow management itself, e.g., by autonomously deciding which individual agents resp. services are responsible for realising specific complex activities, without effect on the overall workflow goals themselves. This decentralized middleware for the integration of decentralized, self-organizing processes among software agents was already presented in [**?**].

The remainder of this paper is structured as follows: The next section gives an overview of related work; Section **??** describes the proposed framework for realising automatic service selection and execution in market-based business applications. Section **??** presents parts of a proof of concept implementation of this framework and reports on a case study to show the applicability of the approach before Section **??** concludes the paper with a brief discussion on future work.

2 Related Work

Service-Oriented Architectures (SOA) provide a paradigm to leverage business integration and to implement loosely-coupled distributed applications. In such an approach, the *dynamic binding* of services is perceived as an integral part of an SOA since it facilitates a loose coupling between services and hence fosters reusability and dynamic adaptation to changing environments. Service binding assembles two different aspects, *service discovery* and *service selection* [**?**]. *Service discovery* describes the procedure of locating available services matching given functional demands. In contrast to that, *service selection* deals with the problem of choosing one service from a set of suitable services. Often, service selection is done by incorporating non-functional requirements, defined by the service requestor [**?**,**?**].

However, since most approaches require the service provider to declare the offered non-functional properties, the consumer has to rely on these propositions. Solutions to this problem incorporate *trust models* in order to rate the reputation of a provider. Vu et al. [**?**], e.g., propose a probalistic framework for decentralized management of trust and quality while incorporating the credibility of observation data, the multi-dimensionality and subjectivity of quality and trust. Advanced approaches for service selection use a *utility function* in order to enable service consumers to flexibly differentiate between important and less significant non-functional properties. Approaches such as the work of Hang and Singh [**?**] provide frameworks which are able to optimize these utility functions.

Regarding dynamic pricing in markets, there has been also work on service selection on behalf of market-based negotiation protocols such as auctions. Wellman et al. give an overview [**?**] of the several bidding strategies used in the international Trading Agent Competition, a scenario where agents contend for flights, hotel rooms and entertainment. Similar to the assumptions in our work, prices are set dynamically by the market participants. For every product type, there is a different auction type with different properties, such as auction setting (e.g. combinatorial or multi-auction), simultaneity, price predictability, auction length. Hence, the approaches of the bidding agents differ enormously.

Lamparter and Schnizler [**?**] propose a market-based architecture for trading Web Services. However, they focus on the semantic description of services and bids with ontologies. Service offers and service requests are converted by a preprocessor in order to facilitate the syntactic matching of bids. This allows for the use of existing implementations of the demanded multi-attribute combinatorial double auction.

He and Liu [**?**] suggest to use software agents in order to realize a market-based service selection framework. However, the resulting framework is nevertheless rather inflexible, since it is limited to the Contract Net protocol and only few non-functional properties are used.

Borissov et al. [**?**] propose an automated bidding system for the allocation of grid-based services. Market platform and bid framework are strictly separated and make use of agent technologies, e.g. for negotiation resp. communication. The *BidGenerator* automatically performs bids at the market, which allows the

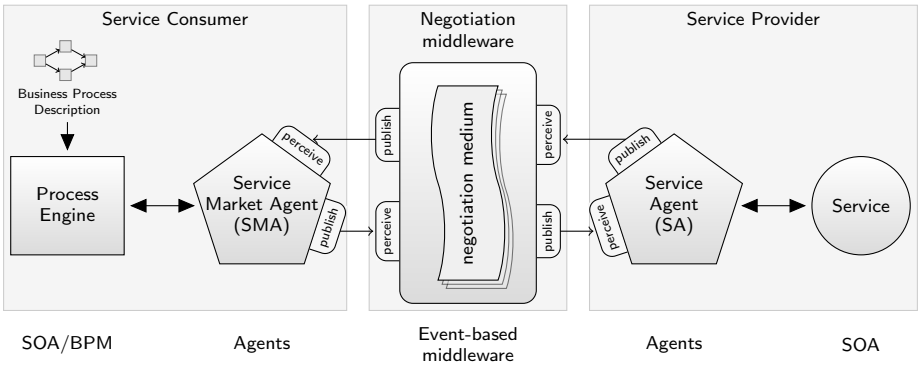


Fig. 1. Architecture of the Negotiation Framework and the utilized paradigms

usage of two negotiation types. Therefore, BidGenerator implements several bidding strategies, which can be used by the customer in order to obtain a resource in the grid.

In summary, the agent paradigm is well suited in order to develop autonomous software components as they are useful for participation in market-based scenarios. Well-established technologies used in agent frameworks, such as negotiation and self-organisation [**?**], can facilitate the development of according applications. However, there are comparatively few frameworks for market-based service selection, which make use of these technologies. The agent-based approaches introduced in this section, however, do not provide support for advanced service-selection mechanisms, such as trust models and business process integration. Therefore, the next section proposes a flexible agent-based service-selection framework that facilitates the implementation of market places using different auction types. It supports the integration of service providers and service consumers into the market with respect to their own bidding strategies, utility functions and trust models, and the overall integration in a business process management system.

3 The Service-Selection Framework

This section presents the proposed framework for an automatic handling of service selection and service execution for market-based business domains. The basic idea of this framework consists of providing a blueprint that contains the necessary components and defines their structure and kind of interaction to achieve the autonomous management functions mentioned in Section **??** rather than proposing and developing new negotiation protocols, utility functions etc. Figure **??** depicts the components of the framework and their structure on an abstract level. It shows that this approach basically uses the SOA approach to build (distributed) applications and takes the *Agent* paradigm [**?**] to enrich the management of these applications with proactive and autonomous capabilities.

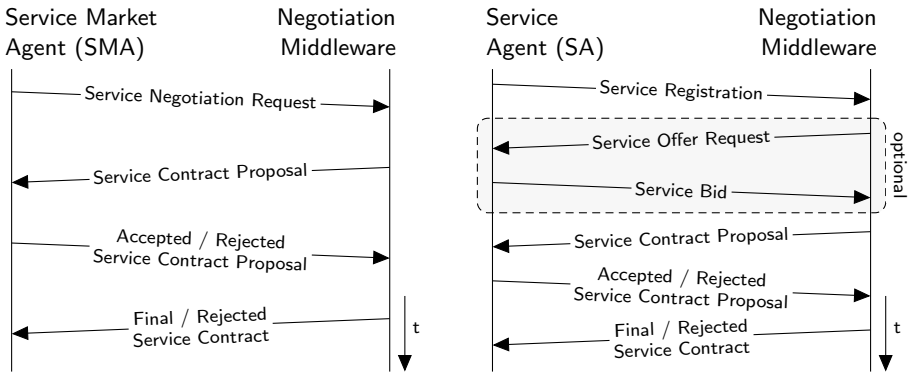
At the beginning of the *service selection* phase, the framework expects a business process description which contains the required services and describes the logical-temporal dependencies among them. This description can be provided by applying, e.g., the *XML Process Definition Language (XPDL)*. It is then processed by a process engine which identifies all needed services and sends for each of them a request to the *Service Market Agent (SMA)*. In addition, this request does not only contain the service type to be found but also some other optional properties like a *utility function* that contains certain concerns regarding time deadlines, fees, quality of service parameters and so forth. Also, the request may specify a certain negotiation (bidding) protocol to be used. For each request, the SMA sends out a *service negotiation request* message using a *negotiation middleware* that contains implementations of various negotiation protocols. Then, it depends on the type of negotiation protocol how an appropriate service provider is selected. In order to perform the negotiation task the negotiation middleware additionally has to process *service bid* messages sent by *Service Agents (SA)*. These agents act on behalf of service providers and try to find contractors that fit best. In summary this approach consists of two agent types, i.e. SMA and SA, that try to find appropriate service partners using a middleware with different negotiation protocols. Details of these components are presented in the following subsections.

3.1 Negotiation Middleware

The aim of the negotiation middleware is to propose an infrastructure component that facilitates the reusability and modularity of negotiation protocols and that additionally enables the *parallel execution* of different negotiation protocols at runtime. Thereby, the negotiation middleware utilizes the approach of *coordination spaces* [**?**], that facilitates a clear separation between computation and coordination. Whereas computation denotes the core functions of a component, coordination can be seen as "managing dependencies between activities" [**?**, p. 90]. In consequence, coordination spaces promote an approach of easy changing the way dependencies are managed among components. This is achieved by a layered approach that contains interchangeable coordination media. In conclusion, different coordination media provide different ways of interdependency management.

From this perspective on, negotiation can be seen as a special type of coordination. Thereby, negotiation can be realized using coordination media that provide implementations of negotiation protocols like the pure ContractNet or extensions of this protocol [**?**] as well as auctions like Dutch, Vickrey etc.

In order to achieve a loose coupling between the SMA, SA and negotiation media the negotiation middleware uses an event-based architecture. Each of the afore-mentioned components implements a generic publish/subscribe interface which enables the components to publish and perceive events of interest. Therefore, the negotiation middleware uses asynchronous communication which is especially suited for distributed systems to ensure a reasonable performance



(a) SMA and negotiation middleware (b) SA and negotiation middleware

Fig. 2. Exemplary communication pattern

level. Another advantage of this approach lies in its flexibility regarding the communication patterns between the single components.

Figure **??** depicts proposals of patterns that have been implemented (cf. Section **??**) for evaluation purpose. The first one (cp. Figure **??**) shows the messages sent between an SMA and a negotiation medium. It starts with a message to initiate the negotiation process containing all information necessary for the negotiation medium to process. As soon as the medium has found an appropriate service provider it sends a contract proposal to the SMA. Now, the SMA can decide to accept or reject this contract. The SMA may decide to reject the proposal if it has already received an appropriate offer from another negotiation medium. Depending on the answer of the SMA the negotiation medium terminates the selection process for this particular service type with a final message that states whether the contract was closed or cancelled.

The proposed communication pattern between SA and negotiation medium is quite similar to the afore described one but has one important difference which relates to the question where to place the negotiation strategy of the SA. Thereby, the negotiation strategy expresses the behaviour of an SA and especially determines the way service bids are computed. One possibility is to place the strategy into the initial *service registration* message that is sent at the beginning. Then, the negotiation medium is the only place where service providers and service consumers interact according to a negotiation protocol like the Contract Net Protocol. This approach has the benefit of a high efficiency since the negotiation process is handled inside the medium and does not require further communication with SAs. On the other hand this approach has the drawback that SAs have to disclose their bidding strategy to a third party (negotiation medium). For certain service provider this might not be appropriate. Therefore, Figure **??** depicts an *optional* part of the communication pattern that targets this issue. When the negotiation medium receives a service negotiation request

sent by an SMA requesting a certain service type, it searches for all SAs that have registered at the negotiation medium for this particular service type. Then, the negotiation medium sends out a *service offer request* to these SAs. Now, the SAs can submit an offer without the need to disclose their strategy. Therefore, proposals are sent back to the negotiation medium using a *service bid* message. Depending on the type of negotiation protocol this optional protocol phase may be iterated several times. Most often iteration is needed if the proposals do not meet the requirements of the SMA. The last part of the communication protocol is equal to the protocol between an SMA and the negotiation medium. It gives the chance to the selected SA, i.e. the SA with the most appropriate bid, to close or reject the service contract (with the service consumer).

As mentioned at the beginning, the framework can implement arbitrary communication patterns. The previous paragraph has presented one possible pattern to illustrate the potential of the framework. The next subsection will present more details regarding the SMA and the SA. It will explain their functionality and show how they can be tailored with respect to certain requirements.

3.2 Two Negotiation Proxies: SMA & SA

In order to provide reusable components for the management of automated negotiations in service-based systems, the two agent types SMA and SA are proposed. This approach promotes a clear separation between the core concepts of SOA and the ability to dynamically negotiate contract partners in the domain of market-based pricing. It enables services as well as processes to participate in negotiations by specifying their requirements and preferences in a declarative manner but without the need to implement these functions by themselves. In order to encapsulate the *negotiation functions* that are needed, two different components have been designed: SMA & SA. Whereas the first one is in charge of service consumers, the latter one deals with service providers. As they act on behalf on another component they are also named *negotiation proxies*.

The two negotiation proxy types themselves consist of reusable subcomponents, i.e. *capabilities* [**?**], which can be divided into mandatory and optional ones. An SMA needs a *Service Offer Capability* and an *Execution Caller Capability*. The first one is in charge of handling the selection phase whereas the latter one takes care of the execution phase. Complementary, an SA has a *Service Supplier Capability* and an *Execution Service Capability* which correspond to the afore mentioned functions of an SMA.

Therefore, the proposed framework can not only be used for the automated management of the service selection phase but also for the dynamic management of the service execution phase. This aspect is achieved by a collaboration between the SMA and the process engine. Once the process engine has started the execution of the process the SMA is contacted each time a service needs to be executed. Now, the SMA is responsible for requesting the selected service for execution as well as dealing with failures. If the selected service is not available the SMA initiates a new service selection process and tries to find an appropriate substitution for the service. In order to detect service breakdowns happening at

runtime the SMA uses a heart beat protocol that determines if a service is alive. Again, if a failure is detected, a new service has to be selected. Additionally, it might happen that a *committed* service provider, i.e. a service that has *signed* a contract with a process, decides to break the contract and to participate in another process, e.g. since it gets a better reward. Then, the SMA acts *proactively* and tries to replace this service in order to avoid a failure when this service has to be executed.

Beside these basic functionalities, the two proxy types can be extended to deal with optional aspects of negotiations like ensuring confidentiality and anonymity among contract partners or using implementations of *trust models* [**?**]. Thereby, this paper does not understand trust from a computer security perspective on. Rather, it uses trust to define and measure the *reliability / reputation* of (possible) contract partners. Reliability tries therefore to quantify the confidence that a stakeholder A has with respect to another one B that it will deliver a service according to a signed contract. Therefore, trust can be seen as an additional important criterion which has to be evaluated in a market-based environment. The interplay of service selections, based on past experiences, and the subsequent adjustment of reputation values leads to an independent dynamic process (cf. [**?**] for a systemic evaluation). Whereas *direct* service selection criteria, such as fees or time deadlines, can be easily evaluated, this does not hold for trust which is an *indirect* criterion and therefore more difficult to measure.

The actual configuration of the proxies for a certain application is performed at design time using an XML configuration file. For both of the proxy types a separate XML Schema has been developed. For the SMA this schema requires following aspects to be specified: an ordered preference list of negotiation media to use, a utility function that is used to evaluate proposals (with respect to potentially conflicting goals) and a deadline for the negotiation process. Optional aspects can be specified as name value pairs. The schema for the SA specifies following aspects: a list of negotiation media to use, a service offer containing the fees and execution period and a negotiation strategy or a reference to the SA in order to avoid disclosing the strategy. These configuration options allow a flexible reuse of the negotiation proxies. In the following section, the practical realization is introduced and the overall approach is evaluated.

4 Implementation and Evaluation

This section describes the prototype implementation of the proposed negotiation framework. Furthermore, a case study on an imaginary "computer manufacturer" is briefly presented in order to demonstrate the applicability of the approach. The framework implementation utilizes the *Jadex Agent Framework* [**?**]. Besides core functionalities required for the execution of *Multi Agent Systems (MAS)* it has the capability to perform (automated) simulation experiments as well as to model and execute processes. Therefore, it is capable of handling process descriptions using the *Business Process Modeling Notation (BPMN)*. Hence, Jadex is well suited as a foundation to realize the negotiation framework.

4.1 Implemented Components

The framework prototype provides reference implementations of the two agent types SMA & SA. Each agent type utilizes two newly provided capabilities for the handling of the service selection phase as well as for the execution phase (cp. Section **??**). Furthermore, the framework prototype includes a negotiation medium that implements a Contract Net Protocol. The design of this protocol is inspired by "the way that companies organize the process of putting contracts out to tender" [**?**, p. 156] but it has also been used in MAS for distributed problem solving. Finally, the prototype utilizes the *Jadex BPMN Engine* and therefore contains reference implementations for all mandatory components.

Beside this mandatory components the prototype offers some optional extensions which enrich its functionality. One targets the chronological order of the phases of a business process. Usually, the selection phase is followed by the execution phase. This might not be appropriate for all types of business processes. Especially processes containing many tasks with a long duration require a different handling since they may face two problems. On the one hand, the selection phase may take too long since there are many tasks which require a negotiation in order to find an appropriate service provider. On the other hand, the dynamic environment of the business process leads to a situation where contract partners may (dis-)appear at any time. Therefore, it is questionable whether it is always reasonable to negotiate contract partners a long time before their service is executed. Rather, the selection of service providers should happen close to their execution in the business process. One possibility offered by the framework in order to cope with these challenges is to relax the strict separation between the selection and execution phases. Tasks of the business process can be annotated with a statement which denotes when a selection process should be initiated for the respective task. Then, the process engine knows which tasks are required in order to start the execution of the business process and which tasks can be selected later. This leads to an approach that is more appropriate for dynamic environments.

Another provided extension targets the issue of trust between contract partners. The extension consists of a *trust capability for service providers, service consumers* and a *trust medium*. The first component observes the execution of service providers and logs in order to detect whether a service request was executed successfully or not. The result is published via the event-based trust medium. In this prototype, the trust medium has the function to transmit messages between service providers and service consumers participating in a business domain that incorporates the usage of trust models. Service consumers use the transmitted logs to compute the trust of possible contract partners. Thereby, they may apply different ways to compute the trust since the received messages represent events without dependence on a particular trust model. For example, service consumers may store the logs in a history and use a simple aggregation function to compute the trust level. The modeling of the aggregation function may be inspired by the process of forgetting of the *human mind* and lead to an exponential function [**?**]. Nevertheless which evaluation function is taken, the

type of service provider	characteristic of fees	characteristic of duration	characteristic of trust
cheap and unrealiable	0.4	0.5	0.999
normal and stable	0.5	0.5	0.4
expensive and reliable	0.7	0.5	0.1

Fig. 3. Characteristics for different agent types

computed level of trust may be part of the service negotiation request message that a SMA sends to the negotiation medium. Then, trust can be used as a criterion for the ranking of bids. Therefore, this approach offers the possibility to realize business markets using trust in a convenient way and it may lead to a situation where service consumers only take bids into account from providers that incorporate trust.

Finally, the framework has an additional component which allows the *automatic* execution of simulation experiments of the implemented application. Besides logging and visualization functions this component allows an easy evaluation of the effects of different bidding strategies. Therefore, the integrated simulation and evaluation component offers valuable support for the development of new components and strategies and it has been used to conduct the simulation experiments presented in Section **??**.

4.2 Configuration of the Case Study

In order to prove the proposed concept and implemented prototype, a case study has been conducted that exemplarily shows the utilization of the dynamic market-based framework. It allows developers to focus on core aspects of the application domain by relieving them from dealing with low-level aspects related to the service selection and the business process execution in a highly dynamic environment. The conducted case study consists of an acquisition process which can be seen as a sub-process of a supply-chain process of a fictive computer manufacturer. This manufacturer defines the logical-temporal order of tasks of the buying process and expects the negotiation framework to manage the execution of the process automatically – including dealing with service failures. For the ease of evaluation the buying process consists of three sequentially ordered tasks that target the acquisition of hard-disks, keyboards and RAM.

In order to enable an automatic management of the business process, the service consumer (computer manufacturer) needs to express his preferences w.r.t. conflicting objectives like fees, service duration and trust. One common approach for that is to apply a *cost-effectiveness analysis* [**?**] and as a part of it to define a *utility function* that values each objective. Mathematically, a utility function can be defined as $U(x) = \sum_{i=1}^{n} x_i \cdot w_i$ where w_i expresses the weight of an objective x_i. Such a utility function is then used to configure the SMA which acts on behalf of the computer manufacturer. Therefore, the utility function enables the SMA to rank different bids and to choose the most appropriate one.

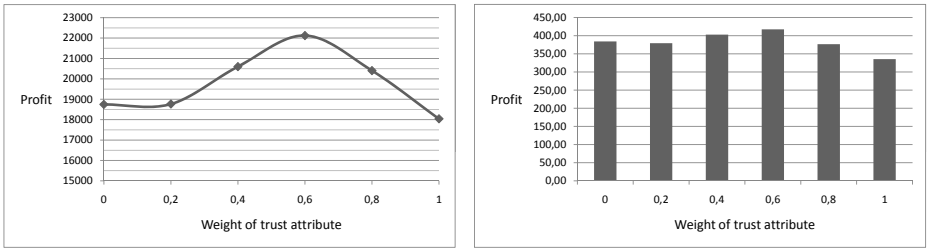
In the same way as service consumers, service providers need to configure their proxy (SA). The case study assumes that the service providers use a static

bidding strategy and will therefore disclose their behaviour to the negotiation medium. Furthermore, the existence of three different *types* of service providers is assumed. These three classes of agent types have specific characteristics w.r.t. following attributes: fees, duration and trust (cp. Figure **??**). In turn, these characteristics influence the bids of service providers. For example, the fees characteristic is used to calculate the fees part of an SA's bid using following function: $f(SA) = \textit{average fees} \cdot (0.5 + SA_{\textit{fees characteristic}})$. Assuming an average fee of €1000 this leads to following bids regarding the fees attribute: €900 (cheap service type), €1000 (normal service type) and €1200 (expensive service type). The same type of strategy is also used to compute the duration part of a bid. Whereas the semantics of the attributes *fee* and *duration* are self-explanatory the semantics of the attribute *trust* requires an explanation. For a service provider the trust attribute denotes the probability of a *service blackout*. A service blackout has two consequences. First, the service provider will not be able to participate in the bidding process (selection phase). Second, the execution of the services that the service provider offers will fail at the moment when a blackout appears (execution phase). The blackout itself is modeled using a function with an exponential distribution. Thereby, the function computes the time between two blackouts of a service provider and takes into account the trust characteristics of the service provider. The higher the trust attribute is, the lower the time is between two blackouts. Then, all the afore described attributes (fee, duration, trust) and settings (bidding strategy, blackout function) are used to configure different SAs to act on behalf of service providers.

In addition to the modelling of the SMA and the SA the case study includes some configuration aspects that are needed to evaluate the outcome of different business strategy scenarios. In spite of all possible conflicting objectives of the computer manufacturer, *profit* is still the most important objective that determines success (of a business process). Hence, the profit of a business process depends on the costs of its execution. These costs consist of the charges of the service providers as well as of two additional fees: First, the service consumer has to pay a fix amount of money to the negotiation medium for each requested negotiation process. Second, the service consumer may need to pay surcharge if the business process is not finished on time. This charge handles the situation where a process B depends on a process A and A is not accomplished on time. For example, this may happen if A contracts to many services providers with a low trust attribute for the execution of its tasks. These providers have a higher blackout probability and tasks will not be executed on time.

4.3 Simulation Results and Evaluation

Based on the described setting of the case study, the negotiation framework is used to evaluate the relation between *the configuration of the computer manufacturer's utility function*, with a specific focus on the configuration of the trust attribute, and its *profit*. Here, the sum of all weights of the utility function with the attributes fee, duration and trust is always 1.0 and the value of the weight for the trust attribute is altered from 0.0 to 1.0 with a step size of 0.2.



(a) Collected profit of iterated business process execution after 60 time units

(b) Profit per business process execution

Fig. 4. Profit in dependency of the weight of the trust characteristic

Figure **??** depicts the *collected* results (average values) of the simulations. Since the focus is on the influence of trust, the execution of the business process is iterated many times using a trust model that incorporates an exponential function to compute the trust level of potential service partners. In this way the level of trust, i.e. with respect to service providers, can change accordingly to successful or failed service executions. More specifically, Figure **??** shows the average results computed from 80 simulation experiments per setting. Each experiment iterated the execution of the business process for a period of 60 time units. The simulation assumes that the service consumer will gain on the one hand a revenue of € 2200 for a single execution of the business process. On the other hand it has to pay between € 900 and € 1200 for the execution of the tasks to the service providers, i.e. depending on the fee characteristic, as well as € 700 for each negotiation process. The simulation results reveal a maximum of profit with a weight of 0.6 for the trust whereas weights between 0.0 and 0.2 as well as of 1.0 show the least profit. Figure **??** uses the same collected results of the simulations and depicts the average profit per business process execution.

The results of this case study show that *trust* has a significant impact on the profit. Taking trust into account, e.g. with a weight of 0.6, pays off with respect to settings without trust, e.g. a weight of 0.0 for the utility function, and increases the profit up to 22%. Furthermore, it can also be seen that configurations with a high trust have almost the same effect on the profit as configurations with no respectively limited trust. The results can be explained as follows. Configurations with a trust value between 0.4 and 0.8 offer the best compromise between reliability, i.e. successful service executions, and costs and have therefore the highest profit. Other configurations have an imbalance and either result in high costs for re-negotiations due to many service failures or choose most often expensive services which limits the profit. Additionally, Figure **??** reveals that the chosen strategies and configurations of the simulation settings reach about 66% of the theoretical maximum of € 600. Of course, this maximum is only achievable without service failures but it can still be seen as a benchmark to compare the outcome of different management strategies within the negotiation framework.

In conclusion, the case study has demonstrated both the applicability of the proposed negotiation framework and the corresponding prototype. The implementation offers the possibility to select and execute business processes in a market-based environment using dynamic and autonomous management components. Also, the evaluation reveals the demand for simulating different management configurations in order to find optimal settings for service consumers as well as for service providers. Therefore, knowledge about the outcome of different settings is the prerequisite for performing management actions autonomously as proposed in this negotiation framework.

5 Conclusion and Future Work

This work targets dynamic market-based business environments that consist of various stakeholders mostly acting in a selfish way. More specifically, this paper addresses software support for dynamic markets with distributed participants that may (dis-)appear at any time. In order to enable a reliable and adaptable selection as well as execution of services in such an environment, this work proposes a framework which incorporates a service-based system at the core, equipped with autonomous and proactive management components (agents). These components act on behalf of their clients, i.e. service consumers/providers, and are therefore also referenced as negotiation proxies. This enables a clear separation of core application tasks carried out by services and of negotiation tasks carried out by agents. Furthermore, an event-based negotiation middleware is introduced which mediates between the demands of the negotiation proxies. The applicability of the framework is proven by a prototype implementation. Furthermore, this implementation shows the capability of the framework to deal with different types of service failures while ensuring the execution of business processes. The negotiation framework is therefore able to automatically process the tasks of service selection and execution for business processes in a highly dynamic environment. In consequence, this allows for – in major parts – highly autonomous management of complex service compositions as typical for advanced flexible and dynamic business processes.

Future work shall, on the one hand, strive towards providing an autonomic and customizable strategy adaptation component for service providers. The aim of this strategy adaptation component is to enhance the success of service providers, i.e. to increase the profit. Therefore, the component observes the outcome of negotiation processes and tries to improve the biding strategy if it is not satisfying. For example: if an SA has a high rate of getting contracts with a certain price the adaptation component may increase the price. Contrary to that, the component may decrease the price if the SA does not close many contracts with service consumers. On the other hand, it is envisioned to provide additional negotiation media implementations in order to take advantage of the potential of the negotiation framework to execute media in parallel. Then, the strategy adaptation component may also manage the aspect which negotiation media to participate in for the SA autonomously.

Acknowledgments. The research leading to these results has received funding from Deutsche Forschungsgemeinschaft and from the European Community's Seventh Framework Programme under grant agreement 215483 (S-Cube).