Contracting Workflows and Protocol Patterns

Andries van Dijk

Deloitte & Touche Management & ICT Consultants, Postbus 300,
1180 AH, Amstelveen, The Netherlands
anvandijk@deloitte.nl

Abstract. Inter-organizational business processes often involve contracting. ICT solutions for contracting processes must offer high flexibility in changing the structure of the contracting process. This can be achieved by 'process-aware' software components which are configured by an explicit model of the contracting process: the contracting workflow. However, the design of a contracting workflow from scratch is a complex task. We propose a solution in which contracting workflows are composed from standard building blocks and show that protocol patterns for business transaction protocols are a necessity for making these standard building blocks available. Finally, we propose a number of protocol patterns for the negotiation phase in a transaction.

1 Electronic Contracting

Inter-organizational business processes often involve contracting. When one organization buys something from another organization, a distinction between 'products' and 'services' is often made. Although products and services differ in many ways, the question is whether these differences are relevant from the perspective of the contracting process. This question is answered by for instance Normann and Ramirez [7], who state "whether customers buy a 'product' or a 'service', they really buy access to resources". Hence, the authors use the term 'offering' to refer to both 'product' and 'service'. Others, like Merz et al [6], have the same approach when they consider payments and tangible goods as services too. In this paper, we will use the term 'service' as a synonym for both 'product' and 'service'.

Service contracting involves information exchange between partners, for which electronic communication is one of the options. The term 'electronic contracting' was already mentioned by Lee in 1988 [5]. In this paper, we define a 'contract' as 'an agreement between two parties in which the mutual obligations are stated'. Furthermore, we define the term 'electronic contracting' as 'a contracting process in which the communication between parties is performed by electronic means and in which the processes at the involved parties are supported by computer applications.' The term 'electronic contracting' is used for a variety of phenomena. This paper is focused on a specific part of this area, which is demarcated by the following characteristics that define a class of service contracting processes.

- *Loosely coupled organizations*
  We assume a relationship between service clients and service providers where all communication is performed by exchanging structured messages, of which only the data types (static aspects) and constraints on the sequence of message types (dynamic aspects) are mutually agreed. We assume no knowledge of each others business processes for the participating organizations.

- *Buyer side only*
  Electronic contracting of services always involves a buyer (client) and a seller (provider). Although these parties communicate via a common message protocol, they execute different processes. This paper focuses on the contracting workflow executed by the buyer (service client). The seller (service provider) is treated as a black box, of which only the external interface (transaction protocol) is known.

- *Multiple required services, Multiple available providers*
  A contracting process is performed for a business case in the enterprise information system. This paper focuses on the more complex contracting processes where each business case requires N different services to be contracted, for which M different service providers are available.

- *Dependencies between services*
  We assume dependencies between required services that define the order in which services must be contracted. For example, service B must be contracted when service A has been completed (sequential relation). Or, service B must be contracted only if service A could not be contracted (alternative relation).

Apart from a demarcation of the class of processes under consideration, we further limit the scope of this paper by focusing on the dynamic aspects of contracting processes only. Data aspects involved in contracting processes, for instance deriving the details of required services from case data or evaluating a received offer, are not in the scope of this paper. A framework for capturing the data aspects of contracting processes is given by Van Dijk [3].

2 Modeling Technique and Approach

Different approaches have been proposed for modeling of the communication between partners in a buying process. In this paper, we view contracting processes as *inter-organizational workflows* and use *Petri Nets* as modeling technique. This choice is made for the following reasons.

- Workflow management techniques based on Petri Nets have a sound theoretical basis and have been successfully applied to internal business processes like shown by Van der Aalst and Van Hee [1]. Since business processes are becoming inter-organizational increasingly, the application of workflow management techniques to inter-organizational processes is an obvious choice.

- Workflow management techniques have proven to be a good solution for repeating, well-structured and potentially long-running processes. The character of the demarcated class of service contracting processes has many similarities with this kind of processes.

- Workflow management techniques are increasingly integrated in software tools for electronic messaging. Apparently, the market recognizes the usefulness of workflow management in combination with electronic business.

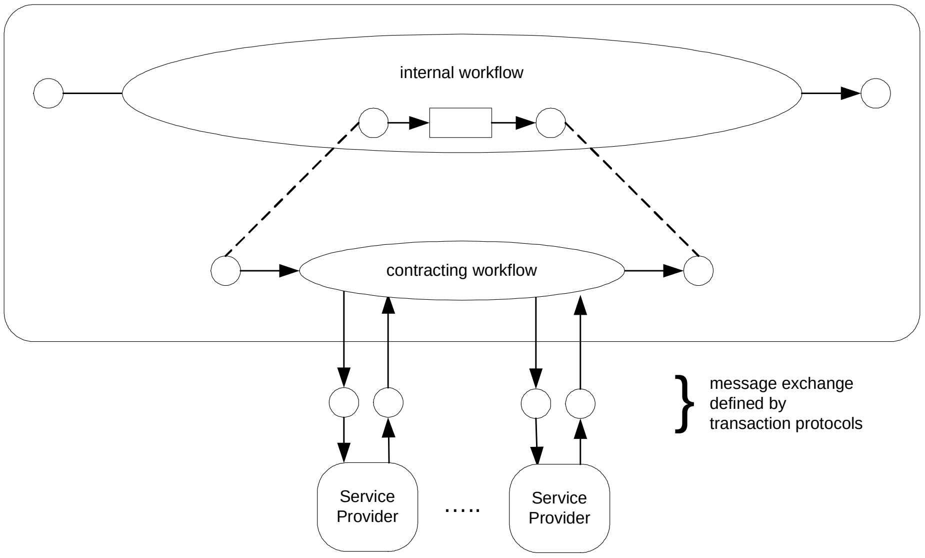

Fig. 1. Position of Contracting Workflow and Transaction Protocols

Clearly, a contracting workflow that involves multiple required services, each of which is controlled by a transaction protocol with multiple messages, can grow to a complexity where it is very difficult for a user to design this workflow from scratch. We therefore propose the following approach.

1. We define a standard high-level workflow structure for the contracting process of a single service (see section 3). The tasks in this high-level workflow need further refinement by replacing it by a sub-net. We assume a library of sub-nets that can be used for that purpose.

2. We start with an empty workflow (start and end place only) and add the standard high-level workflow structure for each different service involved in the process.

3. We add transitions and places to the high-level workflow to model the dependencies between the services (contracting requirements).

4. We add transitions and places to make the high-level workflow sound.

5. We create the final contracting workflow by substituting each task in the high-level contracting workflow with a proper sub-net from the library of sub-nets..

3 A Framework for Contracting a Single Service

A number of frameworks for contracting processes, based on buying products or services from a third party, can be found in literature, e.g. Action Workflow, DEMO (Dynamic Essential Modeling of Organizations) by Dietz [2] and BAT (Business as Action game Theory) by Goldkuhl [2]. These frameworks share the idea that business transactions consist of four phases:

- **Phase 1: Specification**
  In the specification phase, the service client specifies the details of the service to be contracted. In fact, because each service requires a service provider to execute his business process, the specification phase is in its essence the creation of a case token for the workflow in the service providers information system.

- **Phase 2: Negotiation**
  The negotiation phase aims at establishing a contract with a service provider for the specified service. This research focuses on service contracting processes in situations of partial knowledge. This, and the fact that external service providers are often autonomous organizations, implies that a service client can not simply *assign* a task to a service provider but has to *negotiate* with the service provider instead. A contract is established only if there is an offer made by the provider and an acceptance of the offer by the client. The negotiation phase ends either with a contract after which the execution phase starts, or without contract after which the process ends (failed).

- **Phase 3: Execution**
  If a negotiation process resulted in a contract, both service client and service provider will have to fulfill the commitments they entered in the contract. An important aspect of the execution phase is the exchange of status information from service provider to service client, used by the service client to monitor the fulfillment of the contract. The execution phase ends either with the completion of the execu-

tion after which the acceptance phase starts, or it ends with an abortion of the execution after which the process ends.

- **Phase 4: Acceptance**
  The objective of the acceptance phase is to obtain a mutual agreement on the fulfillment of commitments. The service provider declares the fulfillment of his commitments, the service client accepts this declaration and settles the financial obligations towards the service provider. Settlement of financial obligations is however outside the scope of this research. During the acceptance phase, information must be exchanged between service client and service provider. At this point, mutual satisfaction is obtained and the transaction is completed.

This results in a standard structure for contracting one service:

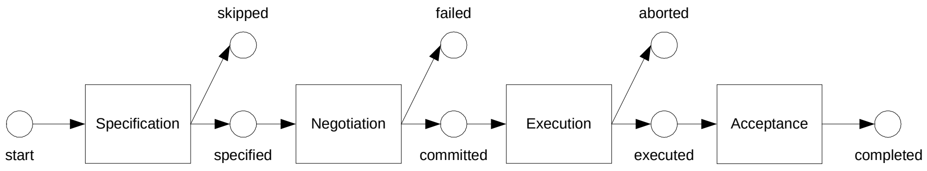

Fig. 2. High-level workflow structure for contracting a single service

4 A Contracting Workflow for Multiple Services

When multiple services are involved in a contracting process, there are always contracting requirements defining the dependencies between the different services. For example: service A can only be contracted after service B has been contracted. Or: when service A can not be contracted, service B must be contracted. This section discusses the rules according to which a contracting workflow for multiple services is composed from the building blocks defined before and from the contracting requirements.

Copy the high-level workflow structure
The starting point for each contracting workflow is a source place ‘start’ and a sink place ‘end’. The first step in creating the contracting workflow is to add the transitions and places (see Figure 2) of the high-level workflow structure for each individual service. This step can be fully automated in the configuration environment of a software component for contracting processes, when the user defines a list of required services.

Model the dependencies between services
The second step in creating the contracting workflow is to add transitions and places that model the contracting requirements, i.e. the conditions under which a token is

produced in the ‘start’ place of the structure from Figure 2. This step can be partially automated when we assume a relative small number of types of dependencies between services. If the user can define the contracting requirements by selecting from the list of services and a list of dependency types, the corresponding changes to the contracting workflow can be made automatically. To illustrate this, we give two examples of dependencies between services and the corresponding representation in the contracting workflow.

Example 1 – ‘B starts when A failed’
In this type of dependency, service B is an alternative for service A. The dependency is modeled by an extra processor that consumes a token from place ‘failed_A’ and produces the token in place ‘start_B’.

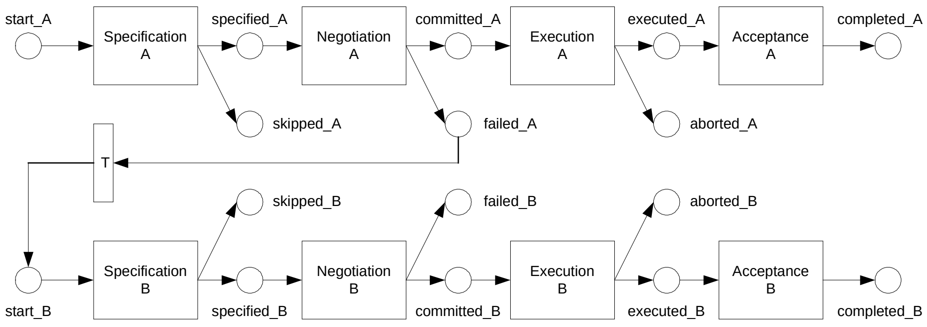

Fig. 3. Example of dependency ‘B starts when A failed’

Example 2 – B starts when A is committed
Another typical type of triggering is when the contracting for a service B is started when another service A has been contracted. This dependency is modeled by duplicating the place ‘committed_A’ into ‘committed_1A’ and ‘committed_2A’ and adding a transition that consumes a token from place ‘committed_1A’ and producing the token in places ‘committed_2A’ and ‘start_B’.

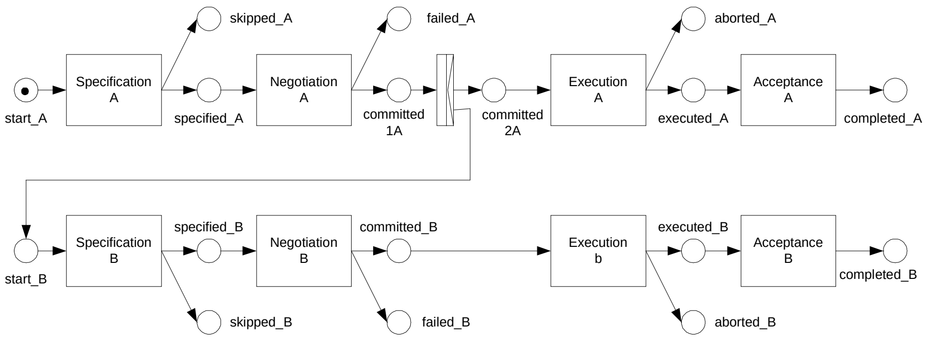

Fig. 4. Example of dependency ‘B starts when A is committed’

Make the high-level contracting workflow sound
We require contracting workflows to be a sound WF-net if we omit the places via which message tokens are exchanged with workflows of service providers. At this point, we have created a structure which is not a sound WF-net. Therefore, the third step in creating the contracting workflow is to add those transitions and places to the high-level contracting workflow that make the resulting high-level contracting workflow a sound WF-net. This can be done by analyzing the distribution of tokens in possible end-states. There after, new transitions and places are added in such a way that in each end-state the tokens that define the end-state are consumed and one token is produced in place 'end'. This task can be automated completely. An example of a sound high-level contracting workflow is given in Figure 5. The example is about a company in which employees have to travel frequently. Each business trip requires two flights to be booked (outbound and inbound). If the employee is not able to travel on one day, a hotel reservation has to be made. Finally, if the employee wants to, a rental car must be made available at the airport of arrival.

Refinement of the high-level contracting workflow
The high-level contracting workflow that we have defined so far is a sound workflow, but is still a high-level workflow where the tasks need further refinement by replacing each task with a sub-net. The structure of the sub-net by which a task in the high-level workflow is replaced depends highly on the transaction protocol that defines the dynamics of the information exchange between service client and service provider. A standard set of sub-nets that can be substituted in the task of the high-level workflow can only exist when there is some type of standardization in transaction protocols. This is why we propose to standardize a relative small number of transaction protocol patterns, on the basis of which we can define the business transactions in specific situations.

5 Transaction Protocol Patterns

Clearly, there is not a single transaction protocol common to all possible service types. Differences in transaction protocols are likely to occur due to differences in legislation, business model, fulfillment processes, etc. However, although we can not present a single transaction protocol for all services, we are able to define *patterns* for transaction protocols. We define a protocol pattern as "a transaction protocol pattern captures the underlying common structure of a set of transaction protocols with different message types but identical dynamic behavior." Since a transaction protocol encompasses the consecutive negotiation, execution and acceptance phases, it can be seen as composed of three smaller transaction protocols, one for each phase. In the rest of this section, we will present patterns for the negotiation phase. Requirements to protocol patterns and correctness criteria can be found in Van Dijk [3].

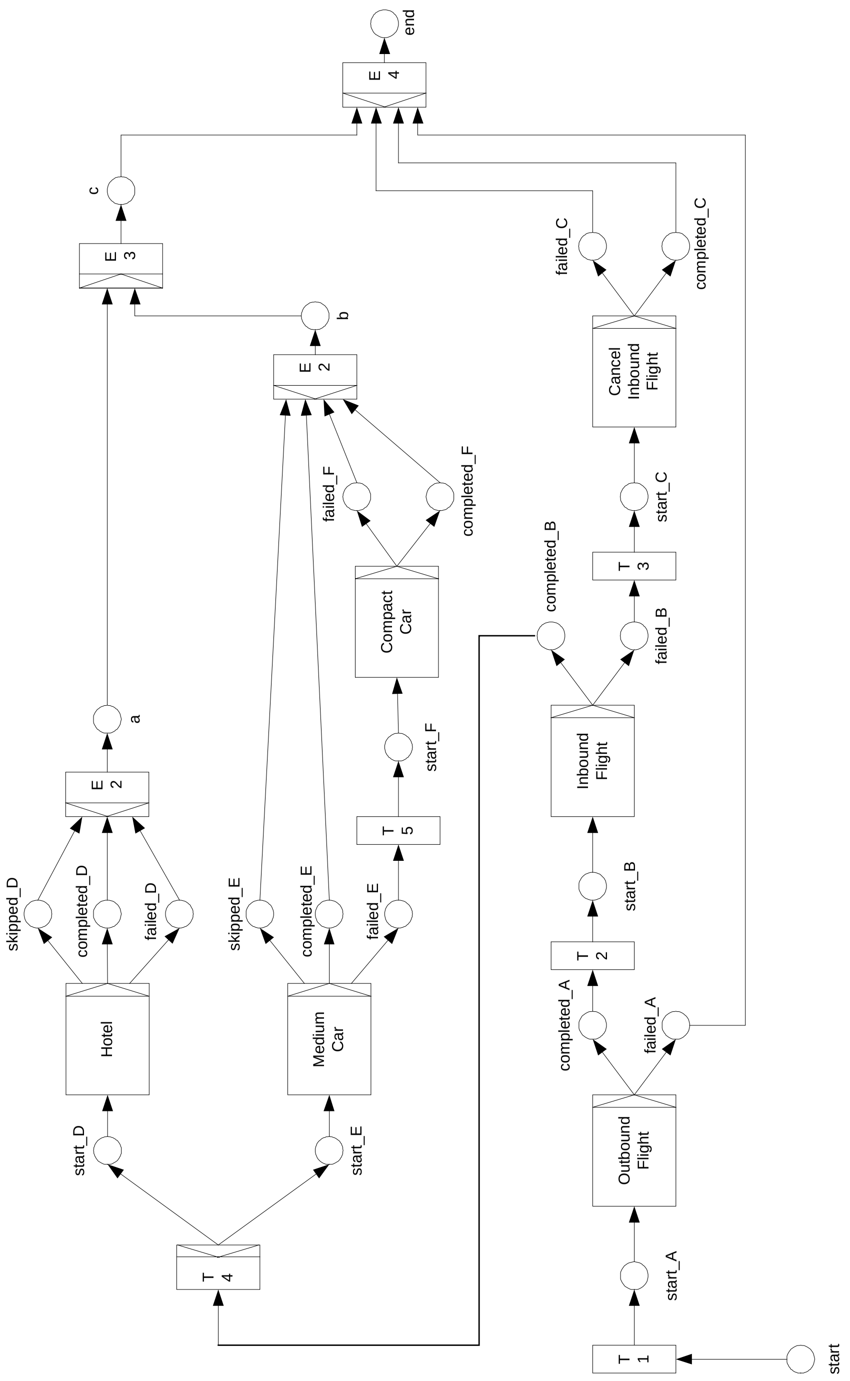

Fig. 5. Example of a sound high-level contracting workflow

Negotiation Pattern: ‘Implicit Accept’

This negotiation pattern is used in situations where there is no explicit response by the service provider to a request made by the service client. Instead, the contract is considered to be established after the request has been made. Clearly, this variant can only be applied under circumstances where the implicit accept is agreed in previous agreements or laws.

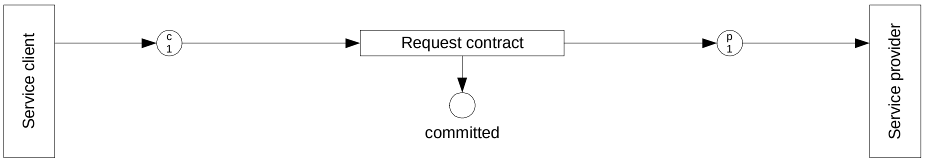

Fig. 6. The ‘implicit accept’ protocol pattern

Negotiation Pattern: ‘Binding Request’

This negotiation pattern is used in a situation where a service client makes a binding request to a service provider, who responds by either accepting or rejecting the request. If the service provider accepts the request, a service contract is established, after which the execution protocol starts. If the service provider rejects the request, neither of the parties has a commitment to each other and the transaction ends.

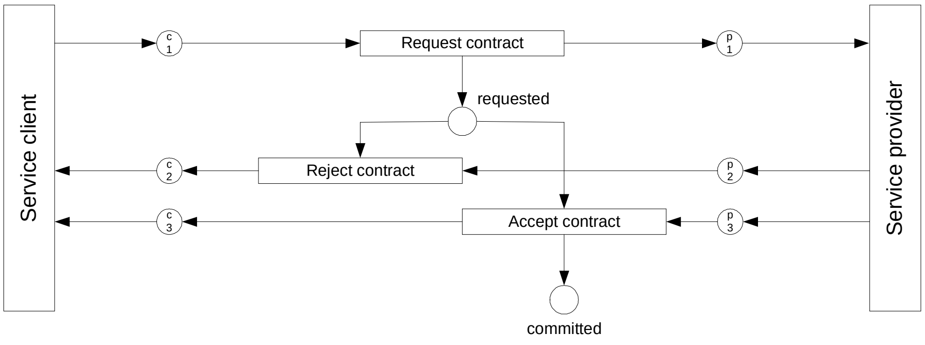

Fig. 7. The ‘binding request’ protocol pattern

Negotiation Pattern: 'Single Binding Offer'

Instead of requesting a contract from a service provider directly, a service client can also request an offer from a service provider. Offers can be binding or non-binding. This negotiation pattern is based on a single binding offer given by the service provider to the service client. When the service provider receives a request for an offer, he either responds by sending a notification that he will not make an offer (e.g. because he is not able to fulfill the request) or he responds by sending an offer message. When the service client receives an offer, he will either accept the offer after which a service contract is established and the execution phase starts, or he rejects the offer after which neither of the parties has a commitment to each other and the transaction ends.

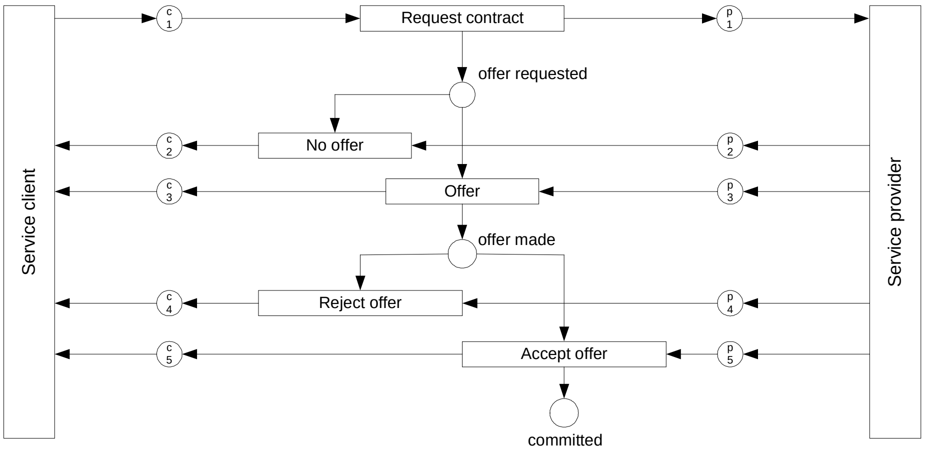

Fig. 8. The 'single binding offer' protocol pattern

Negotiation Pattern: 'Single Non-binding Offer'

An extension to the 'single binding offer' pattern emerges when the service provider sends a non-binding offer instead of a binding offer. This leaves the possibility that after the service client accepted the offer the contract can still not be established, e.g. because the resources required for the fulfillment have been exhausted in the period between sending the offer and accepting it. The pattern is equal to the 'single binding offer' pattern, but has two additional message types that can be received by the service client after he accepted the offer. The confirm accept message indicates that a service contract has been established and the execution phase started. The reject accept message indicates that no contract could be established after all which ends the transaction and leaves both parties without any obligation towards each other.

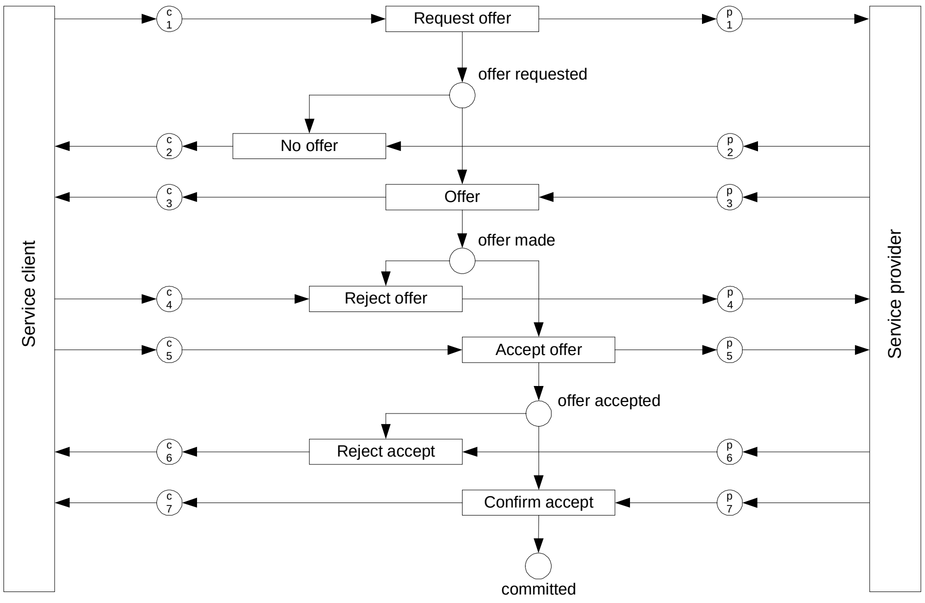

Fig. 9. The 'single non-binding offer' protocol pattern

Negotiation pattern ‘Multiple Binding Offers’

An extension to the ‘single binding offer’ pattern is to allow the service provider to send multiple binding offers instead of a single binding offer, in order to give the service client alternatives to choose from. The negotiation pattern starts with a request from the service client to the service provider. The provider answers either by sending an offer, or by sending a notification that he will not send an offer. If the service provider sends one offer, he can send an arbitrary number of additional offers thereafter. When the service client received one or more offers, he evaluates them and either rejects all offers by sending a reject offers message or he accepts the offer he finds ‘best’ by sending an accept offer message, which contains a reference to the particular offer he is accepting. If the service client accepts an offer, a service contract is established and the execution phase starts. Otherwise, the transaction ends and leaves both parties without any obligation towards each other.

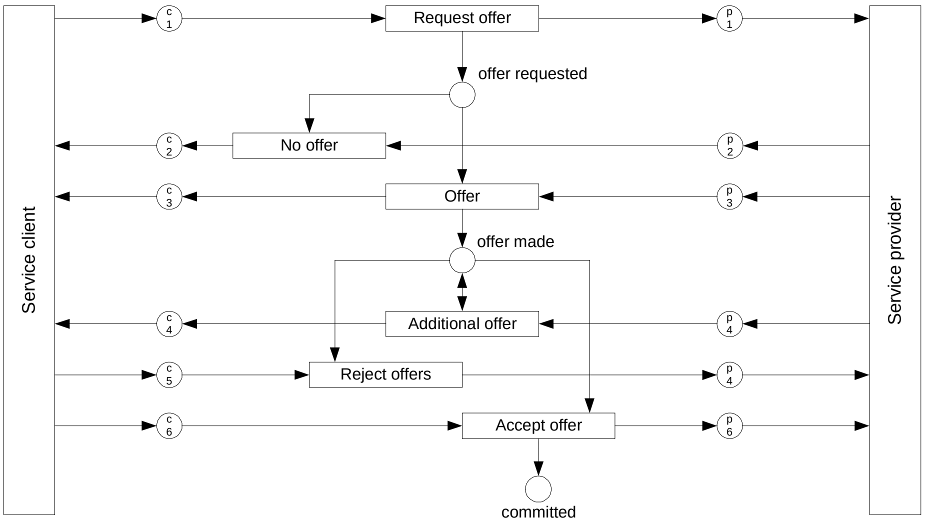

Fig. 10. The ‘multiple binding offers’ protocol pattern

Negotiation pattern: 'Multiple Non-binding Offers'

An extension to the 'multiple binding offers' pattern emerges when the service provider sends non-binding offers instead of binding offers. This leaves the possibility that after the service client accepted an offer the contract can still not be established, e.g. because the resources required for the fulfillment have been exhausted in the period between sending the offer and accepting it. The pattern is equal to the 'multiple binding offers' pattern, but has two additional message types that can be received by the service client after he accepted the offer. The confirm accept message indicates that a service contract has been established and the execution phase started. The reject accept message indicates that no contract could be established. However, the transaction returns to state 'offer made', in which the service client can cancel the negotiation or accept another offer from the pool of offers received from the service provider. In the same state, the service provider can send new offers to the service client.

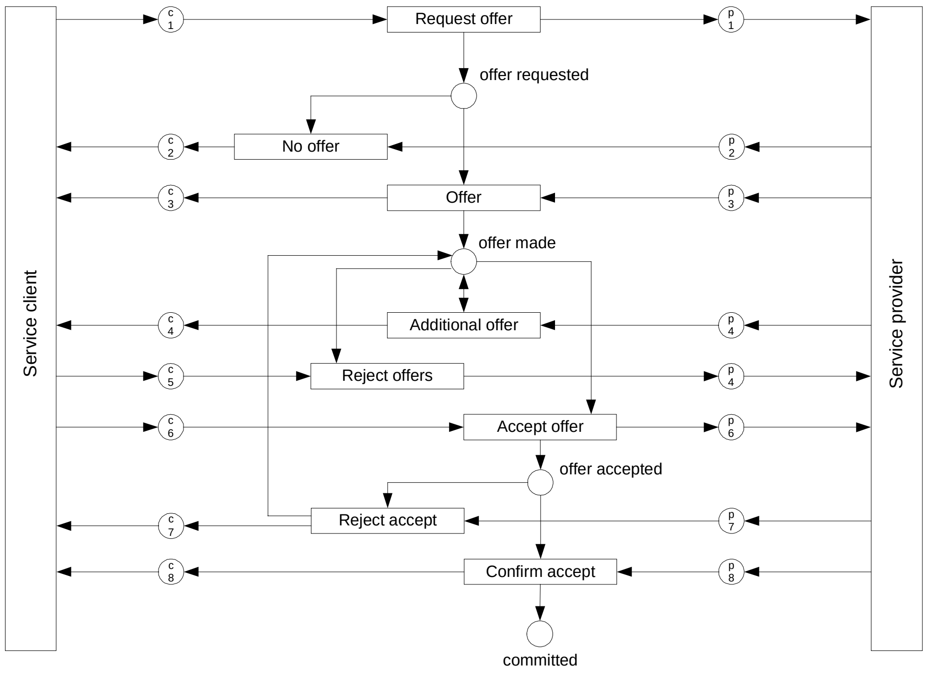

Fig. 11. The 'multiple non-binding offers' protocol pattern

Negotiation Pattern: 'Single Binding Counter Offer'

An extension to the 'binding request' pattern is to allow the possibility of a binding counter offer by the service provider as a third type of response to a direct request for a contract. When the service provider makes a counter offer, the negotiation process enters a state in which the service client can either accept or reject the counter offer and in which the service provider can withdraw the counter offer. If the service client accepts the counter offer, a service contract is established and the execution phase starts. Otherwise, the transaction ends leaving both parties without any obligations towards each other.

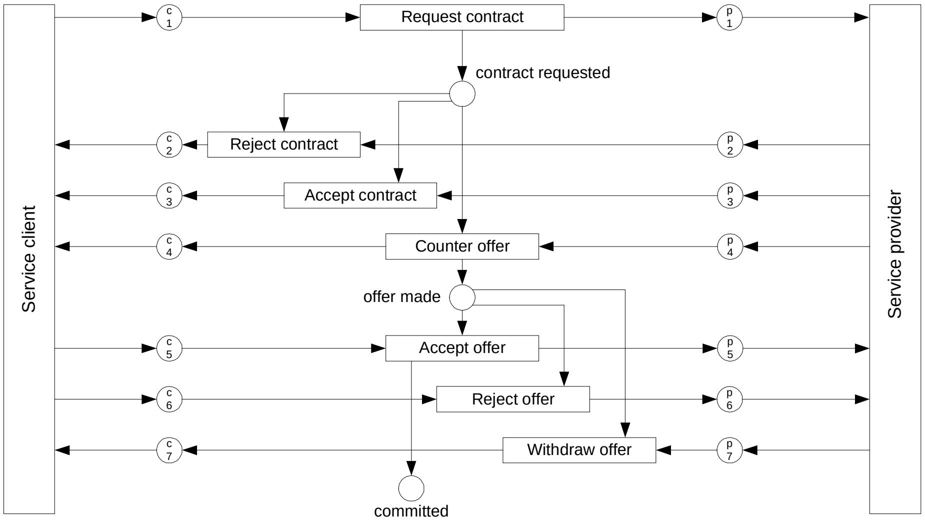

Fig. 12. The 'single binding counter offer' protocol pattern

Negotiation Pattern: 'Alternating Binding Counter Offers'

An extension to the 'single binding counter offer' pattern is to allow the possibility of a counter offer to be followed by a different counter offer made by either the service client of service provider. If a counter offer is made, it replaces all earlier made counter offers. Hence, a maximum of one counter offer can be under consideration at each moment. The party that made the current counter offer can replace it by a different counter offer or withdraw it. The party that did not make the counter offer under consideration can either accept it, reject it, or make a counter offer himself. If a party accepts an offer made by the other party, a service contract is established and the execution phase starts. Otherwise, the transaction ends leaving both parties without any obligations towards each other.

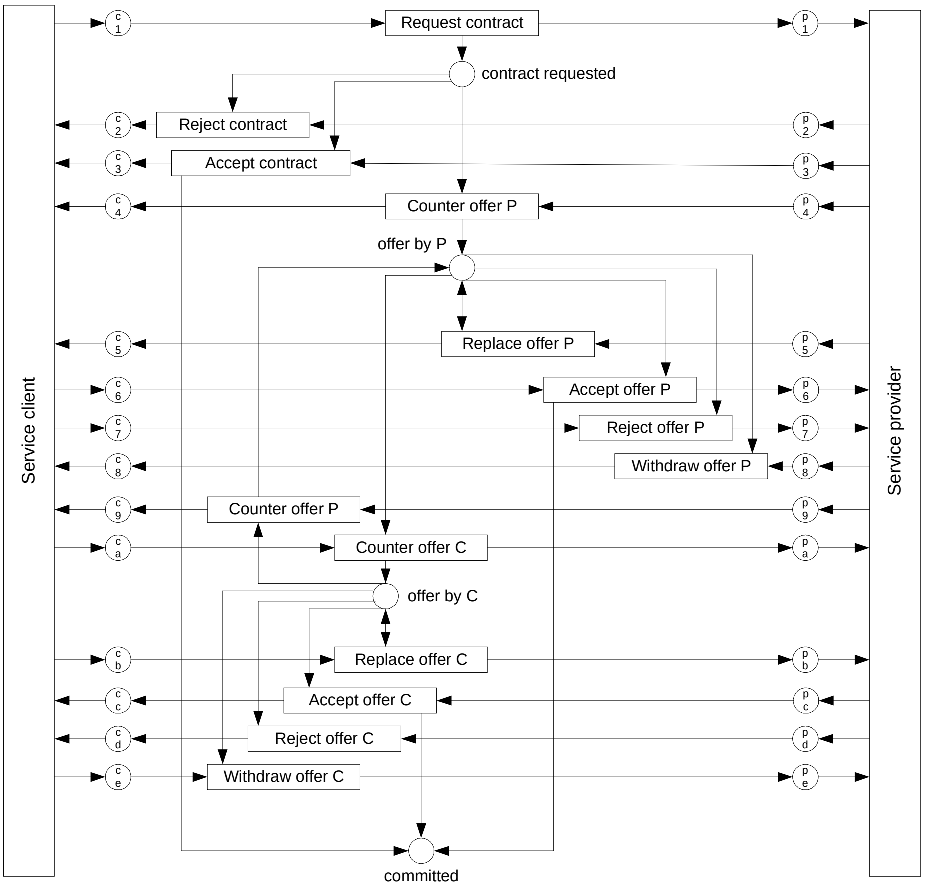

Fig. 13. The 'alternating binding counter offers' protocol pattern

6 Conclusions and Future Research

We have proposed a method for the efficient design of contracting workflows in the configuration process of 'process-aware' ICT components for contracting. The method proposes a standard high-level structure of contracting workflows of which the tasks can be replaced by sub-nets from a library. A prerequisite for this library of sub-nets is a clear standardization of transaction protocols. We propose to standardize a number of protocol patterns on the basis of which specific transaction protocols can be defined. This paper has given an example of possible protocol patterns for the negotiation phase.

Further research has to be conducted to design a set of protocol patterns that is sufficient for the majority of transaction protocols. Mapping of existing transaction protocols to the proposed protocol patterns and the formal standardization process itself are major activities. Another line of research is to extend the Petri Nets defining a transaction protocol with additional quality of service properties (cost of providing services, response times, failure rates, etc.). A transaction protocol extended with these properties can act as an interface agreement between communicating parties. A next step would be the specification of a repository in which a party can publish the transaction protocols that define the external behavior of his services.

References

1. Aalst, W.M.P. van der And K.M. Van Hee, Workflow Management: Models, Methods and Systems, MIT Press, Cambridge, MA, 2001.
2. Dietz, J.L.G., Introduction to DEMO, Samson Bedrijfsinformatie, 1996.
3. Dijk, A. van, The Contracting Agent – concepts and architecture of a generic software component for electronic business based on outsourcing of work, Ph.D. Thesis, Eindhoven University of Technology, 2001.
4. Goldkuhl, G., Generic Business Frameworks and Action Modeling, Proc. of 1st International workshop on Communication Modeling, Springer-Verlag, 1996.
5. Lee, R.M., A Logic Model for Electronic Contracting, Decision Support Systems, Vol. 4, No. 1, pages 27–44, 1988.
6. Merz, M., F. Griffel, T. Tu, S. Müller-Wilken, H. Weinreich, M. Boger and W. Lamersdorf, Supporting Electronic Commerce Transactions with Contracting Ser-vices, International Journal of Cooperative Information Systems, Vol. 7, No. 4, World Scientific, 1998.
7. Normann, R. and R. Ramirez, Designing interactive strategy: from value chain to value constellation, John Wiley & Sons Ltd, 1994.