Patterns for Service-Oriented Information Exchange Requirements

Ayman Mahfouz, Leonor Barroca, Robin Laney, Bashar Nuseibeh
Department of Computing,
The Open University,
Walton Hall, Milton Keynes, MK7 6AA, UK
amahfouz@gmail.com, {L.Barroca, R.C.Laney, B.Nuseibeh}@open.ac.uk

Abstract

Service-Oriented Computing (SOC) is an emerging computing paradigm that supports loosely-coupled inter-enterprise interactions. SOC interactions are predominantly specified in a procedural manner that defines message sequences intermixing implementation with business requirements. In this paper we present a set of patterns concerning requirements of information exchange between participants engaging in service-oriented interactions. The patterns aim at explicating and elaborating the business requirements driving the interaction and separating them from implementation concerns.

1. Introduction

Service-Oriented Computing (SOC) is a software development paradigm that adopts the notion of a "service" as the fundamental unit of building and composing applications. A service is a self-describing high-level abstraction of coarse-grained business capability. Services hide the complexity of the IT infrastructure and the heterogeneity of platforms behind standards-based interfaces.

Services can be published to registries where they can be discovered by potential service consumers, therefore SOC promotes loose coupling between interacting participants. SOC has enabled the creation of electronic marketplaces where enterprises can compete for e-Business opportunities and collaborate electronically via autonomous agents.

Service-oriented interactions are complex in nature. They cross the borders of the enterprise and span multiple independent organizations, each of which has its own processes and internal systems independent from other organizations. Each participant in the interaction has its own logical state, such as data in a database or a legacy system, and physical state comprised of business resources as well as humans involved in the interaction. Furthermore, service-oriented interactions are often asynchronous and long-running, thus over the duration of an interaction the state of each participant may change.

Process-oriented languages, such as BPEL[1], are the dominant way of describing multi-party SOC interactions. Such languages have been criticized for intermixing the business rules driving the interaction with implementation-specific messaging mechanisms in one description[2]. The business requirements of the interaction are concerned only with the content of the information (what), the purpose it is needed for (why), the participant providing/requiring it (who), and possibly the time it is needed/used (when). Business requirements do not normally specify the exact messaging sequence by which information is exchanged (how), which is an implementation concern usually driven by architectural constraints.

Separating out the business requirements from implementation concerns is important because it allows us to focus on elaborating and structuring the business requirements without having to make early decisions about implementation of these requirements in terms of messaging sequences. Furthermore, by establishing a mapping between certain classes of requirements and their typical implementation

mechanisms we can derive an implementation given a set of business requirements and verify that the implementation satisfies the requirements.

In this paper we captured a set of patterns concerning requirements on information exchange between participants involved in service-oriented interactions. The patterns were gathered by examining several examples from the SOC literature as well as some SOC applications. The patterns place the emphasis on the problems and the requirements rather than on low level messaging aspects of SOC interactions and as such they do not fall under the "design" patterns category. It also follows that our patterns do not address SOC realization concerns. Specifically, we do not make the explicit distinction between services in the general SOC sense and Web Services [3] as the realization of SOC on the internet using XML technologies.

The patterns presented are intended to assist in eliciting SOC requirements in a semi-structured manner. Each pattern encapsulates a "piece" of a problem along with the relevant considerations for this type of problem. The considerations associated with each pattern in the catalogue assist the user of the pattern when applying it to a problem at hand in asking the relevant questions to elicit the business requirements. Moreover, we take a step towards building a pattern language by explicating relations that connect the patterns. These relations provide guidance on traversing the requirements space and uncovering more patterns that can be used in elaborating other parts of the requirements.

The paper is organized as follows: In the next section we give an overview of the patterns and their interrelations. Section 3 discusses related work. In section 4 we introduce an example to motivate the work. Section 5 details the patterns catalogue. We revisit the exemplar to demonstrate the application of the patterns in section 6. We conclude the paper and discuss future work in section 7.

2. Overview of the Patterns

Having surveyed several exemplars from the SOC literature, we noticed the recurrence of certain patterns of business requirements involving the information exchange between participants in SOC interactions. The main concern of each pattern is briefed in table 1. We make no claim about the completeness of the set of patterns, which will undoubtedly be refined and expanded as we survey more exemplars.

Pattern	Main Concern
Barrier	Guards an action and specifies (pre)conditions on its execution
Co-location	Two or more resources are to be co-located at a certain time and place for a specified duration.
Correspondence	Relating two pieces of information each owned by a different participant
Deadline	Some information is required for an action before a certain time after which an alternate action is taken
Expiration	Some information will become invalid at a certain point in time (not shown in figure)
Notification	On-state-change "pushing" of information to enforce ***Correspondence***.
Query	On-demand periodic polling of information to enforce ***Correspondence***
Retry	Retrying an action a number of times before resorting to an alternate action
Selection	Choosing from among similar service offerings from multiple participants according to some criteria
Solicitation	Gathering information about service offerings from participants
Token	Issuing a permission for executing an action to other participants

Table 1. Brief description of each of the patterns

Figure 1 depicts the patterns and the relations between them. The patterns are shown in boxes with labeled directed links between them while related concepts are shown without boxes. The relations

between patterns are intended to assist in traversing the problem space. An example of how to interpret and use the diagram goes as follows: A ***Selection*** may require ***Solicitation*** of multiple participants, each of which may require a ***Token*** to participate, where the token implies ***Correspondence*** between the copy that resides within the solicitor and the one that resides within the solicited participants, so the solicited participants may get a ***Notification*** that the token they possess is no longer valid.

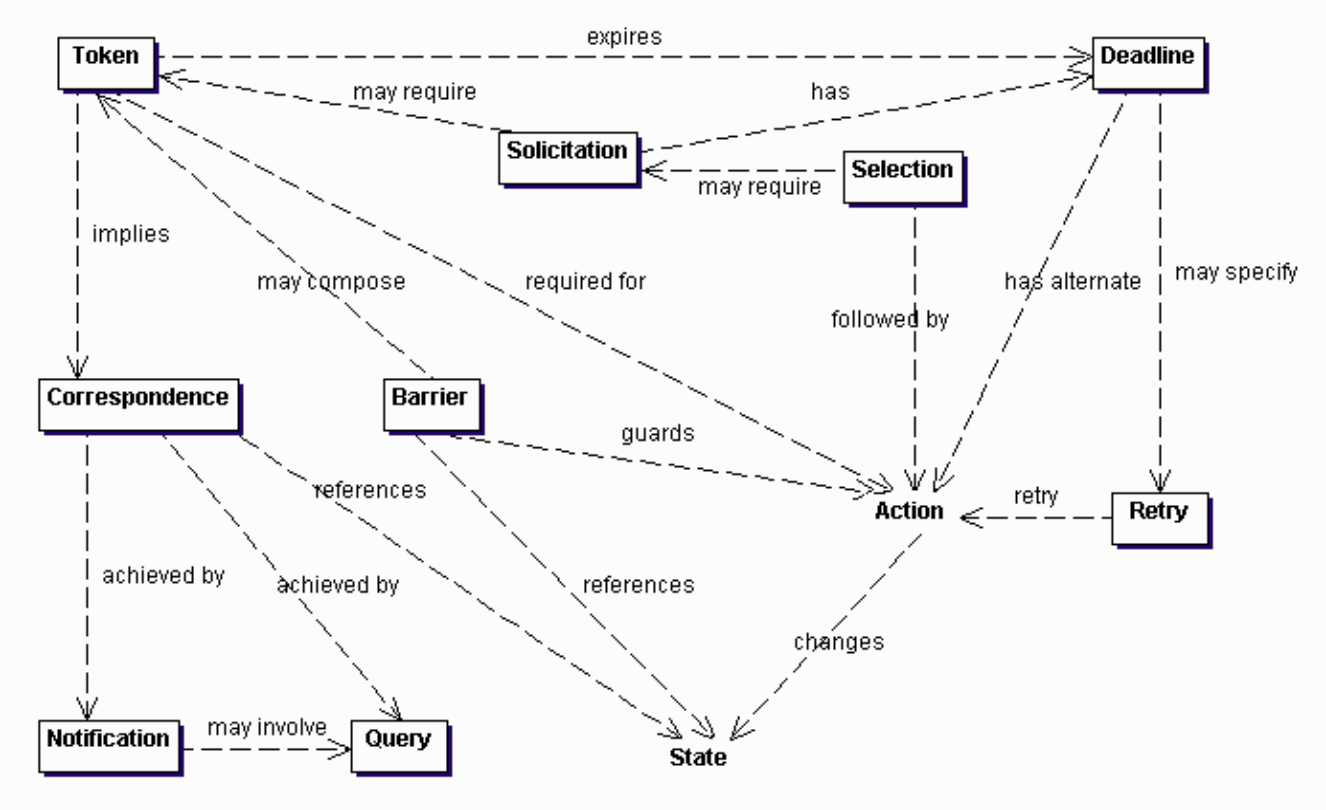

Figure 1. The patterns and relations between them

3. Related Work

Previous attempts to catalogue service-oriented patterns have focused on low-level aspects of service-oriented interactions such as the number of participants, the number of messages exchanged, and the direction of message flow [4], which capture interesting details about the interaction but do not address the business problem driving the choice of an interaction pattern.

The same goes for the integration patterns in [5] which are intended to provide a vocabulary and a visual notation framework to describe integration solutions. The patterns address aspects of a messaging system such as connecting an application to a messaging system, routing messages, and health monitoring. Although the catalogue encompasses an extensive set of patterns, it does not go beyond implementation and design levels.

Property specification patterns (PSP) [6] were used to specify and validate web service interactions in [7]. The patterns specify constraints on the occurrence and ordering of web service operations in a declarative manner amenable to composition. To this end, the PSP patterns aim to replace the typical procedural description of business process languages rather than elevate the description to the level of the requirements behind the process or provide guidance on eliciting such requirements.

There is a void in the literature of patterns that address the business requirements of the information exchanged between participants in SOC interactions (rather than on the design and implementation of the messaging that satisfies these requirements). Our patterns are an attempt to fill part of this void.

4. Motivating Example

The main exemplar that will be used in illustrating the patterns involves a medical provider (MP) which operates a number of hospitals and medical facilities at various locations. Here are some snippets of the business requirements:

- The MP partners with an ambulatory service that transfers patients to the medical facilities. To optimize their service, the ambulatory service has the freedom to choose which medical facility to transfer a patient to depending on the patient location, his condition, among other factors.
- The hospital purchases medical supplies from several vendors most of which provide periodically updated price lists.
- Some patients have medical insurance that covers the cost of their treatment. For some types of insurance, a treatment authorization has to be pre-obtained from the insurance company providing the coverage for a patient.

These high level business requirements need to be elaborated to a much greater detail before even thinking about the sequence of messages to be exchanged between the participants. For instance, there are numerous questions that have to be asked (and answered) about the "treatment authorization" including: What distinguishes one treatment authorization from another? How does the authorization identify the patient? Can the authorization be used more that once (to treat the same patient from the same ailment, for instance)? Does the authorization expire? Can the insurance company cancel the authorization? What if the authorization has already been used to prescribe some medication for the patient?

As can be seen, coming up with these questions is quite a task even for such a small requirement snippet. Our patterns and the relations between them are intended to assist in evoking such questions thereby improving the process of navigating and elaborating the business requirements of SOC interactions. We will refer to the exemplar as the "MP" example thereafter.

5. The Pattern Catalogue

Due to the lack of space, the catalogue presented here details only five of the patterns we have identified: ***Token***, ***Correspondence***, ***Selection***, ***Solicitation***, and ***Deadline***. Our catalogue roughly follows the Alexandrian form as well as other popular template forms [8]. The template comprises context, problem, forces, solution, resulting context (consequences), examples, and related patterns. We illustrate the structure of each pattern using either a conceptual class diagram or an object diagram where it helps distinguish between multiple instances of the same concept. Most importantly, the "Considerations and Variants" section has the bulk of details about the pattern and is intended to be used in generating questions about the fragment of requirements it is applied to.

Since we are presenting patterns about interaction, the participants are part of the "structure" of the pattern and hence we did not include a "participants" section. Also, the "solution" embodied in each pattern is a high level prescription, rather than a specification of an implementation that details a sequence of message exchanges and hence we did not include a "behavior" or "collaborations" section.

5.1. Token Pattern

Context

In a multi-party interaction each participant has its own system that is logically and geographically separate from the other. Certain business rules may dictate that one party should not attempt to undertake a certain action in the course of a business interaction unless some explicit permission is obtained from another party. In a traditional non-electronic business interaction the permission would typically be a signed paper document.

Problem
How should the permission be represented, enforced, and managed?

Forces
- The business rule stipulating that "a permission has to be obtained for the action to be performed" must be enforced.
- The permission has to be conveyed from the issuer to the party that needs it.

Solution
The permission is represented as an electronic "token". The availability of such a token to the party wishing to execute the action means that the party may go ahead and do so, whereas the lack thereof means that the party should not.

Structure
The class diagram in Figure 2 depicts the structure of the Token pattern. The participant providing the permission issues a token that enables the action to be taken by the second participant.

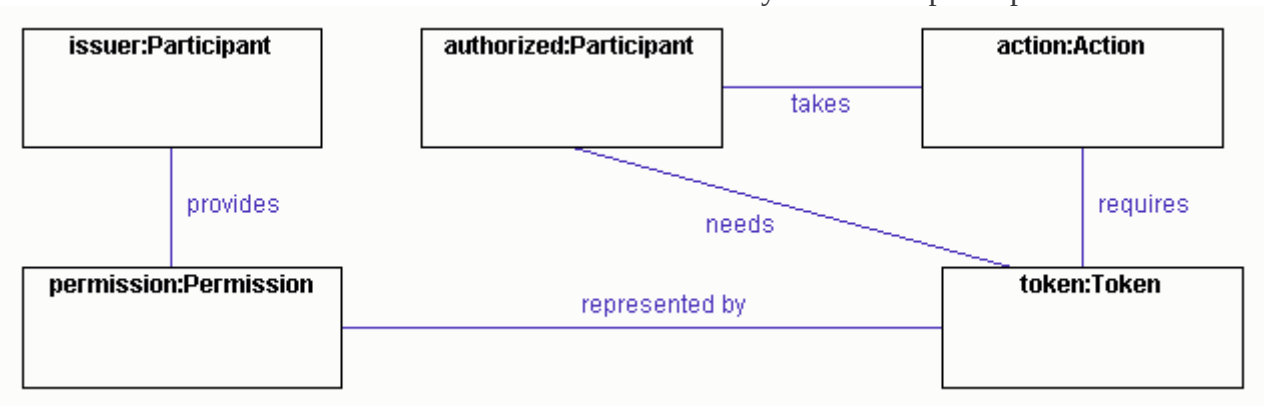

Figure 2. Object diagram showing the Token pattern structure

Considerations and Variants
- ***Identities***: The token typically has information that uniquely identifies it from all other tokens of the same type in a certain usage context. The token also identifies the action it is required for as well as the specific instance of the action. Consider a vendor that issues a "Merchandise Return Authorization" (MRA) so that a buyer can ship back a defective item. The MRA has a unique number that identifies it from any other MRA the vendor issues. The MRA enables the action "return merchandise" which is instantiated for a particular product returned from a particular buyer.
- ***Multiple Required Tokens***: An action may require more than one token. For example, if several parties have to vote to allow the requestor to perform the action or where the requestor is required to request the permission of more than one party. A generalization of the "multiple required tokens" is where a number of instances of different types of tokens are required.
- ***Multiple-Usage***: A batch of tokens may be obtained and stored by the requestor for subsequent use. Consider the example where a wireless provider issues a signing key to a software development company. The key is to be used by the software development company in signing code to be delivered over the wireless network operated by the wireless provider. The wireless provider will typically limit the number of times the key can be used to sign code. The software company will have to purchase another key after the limit on signing attempts has been reached. This can be viewed as obtaining multiple tokens at once or as obtaining a token that can be reused for a specified maximum number of times.
- ***Recyclable Token***: In some cases a token may be reusable over and over for an unlimited number of times. A special case of this is where the token usage lasts for some amount of time during which the token is "locked" and can not be used otherwise. The token can only be used

again after it has been "released". An example is where a service (or a web site) allows only a single session for a particular user. Another example is where a software company sells "concurrent user licenses" for their software.

- ***Action consuming the token***: The action that requires the token may itself consume the token when it executes or an additional action may be needed to consume it. An example of the former is where the token gets consumed when an instance of a concurrent license is "checked out" of the repository. An example of the latter is where the MRA is a token that allows the buyer to take the action "return item", the MRA is not actually consumed until it is entered into the vendor system when the returned item is unpacked.

Consequences

- A Participant can grant another participant a permission by issuing a token.
- The token becomes a representation of the permission that gets transferred electronically between the participants.
- The token and the permission remain two separate and distinct things. The lifecycle of the permission may not exactly coincide with that of the token. For example, the token could be created after the permission is issued.

Examples

- In the MP example the "treatment authorization" is a permission required for performing the action "administer patient treatment". The insurance company provides an electronic form of the permission that can be used by the MP.

Related Patterns

- A Token may often be obtainable through ***Solicitation***.
- ***Correspondence*** between the state of the token representation on the provider side and on the requestor side may have to be maintained. For example, if the provider is allowed to cancel the token this state change has to be relayed to the requestor.
- A token is often associated with an ***Expiration***. For instance, an MRA is only valid for a certain number of weeks from issuance. A software license may also be time-limited and has to be renewed.
- Tokens are central concept in (Colored) Petri Nets [9]
- A token may serve a similar purpose as a "guard" in the authorization pattern [10].

5.2. Correspondence Pattern

Context

In long-running interactions information is exchanged asynchronously between multiple parties over a relatively long period of time. Each participant has its own internal process and internal state independent from other participants. However, the progress of one participant's internal process may cause state changes that should have an effect on another participant's process.

Problem

When each participant has its own internal process and state, how do we relate one participant's process to the process of other participants and determine the effects it has on those processes?

Forces

- Each of the participants has their own internal processes and state that are not shared with other participants.
- The internal process of one participant may cause changes to information that is of importance to another participant.

- The state of a participant comprises both logical and physical state. Therefore, state changes that happen within the realm of one participant may not be immediately available to other participants.

Solution

Establish pair-wise correspondence between the information of interest (information A) at participant A and related information (information B) on the participant's B end. Determine the state changes in information A that are of interest to participant B and should have an effect on the state of information B. The business events that cause changes in the state of information A need to become "shared" events that B gets to know about.

Structure

Figure 3 depicts the concept of ***Correspondence*** between two pieces of information each of which owned by a different participant.

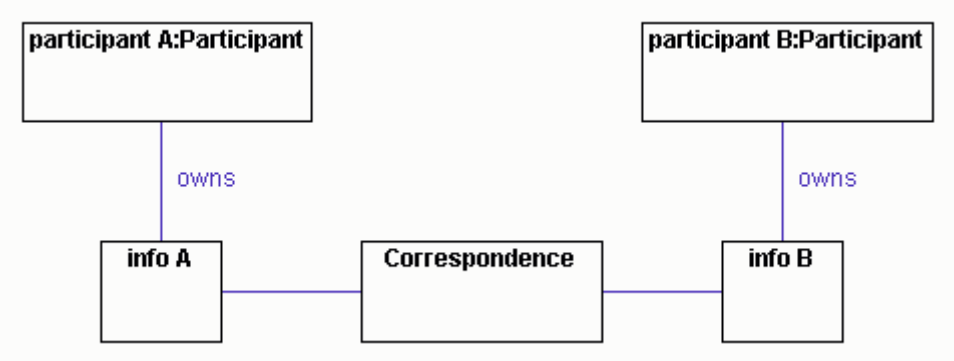

Figure 3. Object diagram showing the Correspondence pattern structure

Considerations and Variants

- ***Propagating changes***: The Correspondence pattern only deals with establishing the relations between the information and determining the required effect that one participant's internal process should have on the other. The actual mechanisms of propagating the changes are the concern of the lower level patterns ***Notification*** and ***Query***.
- ***States and Transitions:*** Conceptually, this pattern is concerned with tying two state machines together by defining "shared" transitions. In other words, correspondence is established between a transition in the state machine representing one participant's process and a related transition that should take place simultaneously in the other participant's state machine. The transitions result in the change of the state of information held at each participant.
- ***Partial Correspondence and Thresholds***: Often only certain changes in the state of an object are of interest to another participant and only those changes need to be shared. For instance, there may be no state on the MP side corresponding to the state where the treatment authorization is "in process" at the insurance company. For numeric state "thresholds" may determine whether a state change is to be shared or not. For example, a stock broker may need to know when the price of a certain stock rises above a specified threshold.
- ***Multiple copies***: The simplest form of this pattern is where each participant keeps his own copy of an object that is being exchanged. For example, a buyer will have a representation of a "purchase order" which assumes states such as: "created", "sent", "confirmed", etc. and a seller will have a corresponding concept of a purchase order that assumes states such as: "received", "processed", "fulfilled", etc. Correspondence between these states can be established depending on the specific requirements of the situations.
- ***Clock***: The state of each participant consists of both logical and physical state. The physical state involves objects from the real world including paper documents, vehicles, humans, etc. Additionally, systems are geographically distributed and networks introduce delays. Therefore, state changes are not instantaneous and information about the new state may not be

immediately available to other participants. In other words, the participants do not share the same clock and the state changes do not happen simultaneously on both sides.
- ***Out-of-date view:*** It follows from the above that each participant may have out of date information about the rest of the world. Therefore, actions taken based on assumptions about other participant's state may later be found to be invalid. We have developed a set of strategies for dealing with this situation that are out of the scope of this paper.
- ***Chaining***: If A corresponds to B and B corresponds to C then A (indirectly) corresponds to C. Determining the effect of A on C can be determined by combining the effect of A on B and the effect of B on C. In other words, correspondence is transitive.

Consequences
- The internal state of a participant can be shared with another participant.
- Each participant can assess the impact of other participants' process on their own.
- Changes to logical and physical state involved in a correspondence has to be tracked in order to maintain consistency between the states of participants at runtime.

Examples
- There are two corresponding representations of a treatment authorization; one resides within the insurance company while the other resides within the MP. The cancellation of the treatment authorization at the insurance company means that the authorization held by the MP is no longer valid.
- The ambulatory service maintains a "preferred medical location" list which maps each Zip code to the location of the medical facility to be chosen for a patient transferred from that Zip code. There is an indirect correspondence between the preferred medical facility and the current workload (number of patients relative to number of doctors) at that facility. For example, the workload at a given facility may become temporarily too high to the extent that another facility should be designated the preferred location.

Related Patterns
- The ***Notification*** pattern and the ***Query*** pattern are concerned with the mechanisms for enforcing the ***Correspondence***.
- The GoF ***Observer*** pattern [11] is typically used to enforce ***Correspondence*** by ***Notification***.
- In the problem frames framework [12] the "Information Display Frame" deals with the correspondence of real time information and its physical display.
- Part of what WSCDL [13] deals with is interaction-based information alignment between state that resides in one "role" with corresponding state that resides in another.

5.3. Selection Pattern

Context
Service-oriented software allowed the creation of open e-marketplaces where potential participants in service interactions present competing service offerings. Other participants can then pick and choose from among competing service offerings that match their needs.

Problem
How does a participant take advantage of the availability of multiple potential participants that present competing offerings?

Forces
- Offerings provided by the competing participants are functionally similar or the same.
- Choosing one participant over another may optimize a certain quality while compromising on another.
- The potential participants and their offerings may change from one interaction to the next.

Solution
A participant selects among multiple candidate providers according to one or more criterion that optimizes certain qualities of the interaction.

Structure
Figure 4 depicts the structure of the ***Selection*** pattern. Several participants can be candidates for selection, from which some may get selected according to one or more criterion each of which may have an associated weight relative to the other selection criteria.

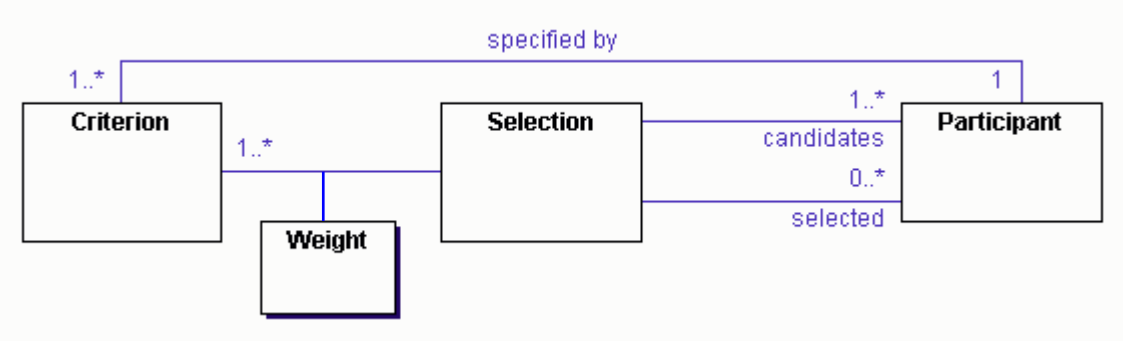

Figure 4. Class diagram showing the Selection pattern structure

Considerations and Variations
- ***Criteria***: The criteria on which the selection is made may be one of several typical criteria:
    - The provider with the most cost effective offering.
    - In the cases where the offers were solicited the selected participant may simply be that whose offer is received first.
    - Where a provider "rating" history is available, the provider with the best ratings score is selected.
- ***Weights on criteria***: The selection may be made based on more than one criterion at once. For example, the criteria could be a composite that takes into account both the cost of the service and its reliability. In such a case weights should be assigned to the criteria in order to make the selection objectively.
- ***Select more than one***: Depending on the nature of interaction, it may be required that multiple participants get selected.
    - If the goal of the selection can be decomposed then a participant can be independently selected to fulfill part of the goal. For example, if the goal is to minimize the total price of a list of items being purchased, then each item may be purchased from the participant that provides the lowest price for that item.
    - Selecting more than one participant may be a form of "fault tolerance". One participant is designated to be the main participants and one or more are selected as backup. In case the first selected participant fails to fulfill their responsibility, the "runner-up" is tapped instead.
- ***Phases***: The selection could be a process that goes through successive phases before a final participant is selected. The candidates are filtered out in each phase where each phase may have different (or additional) criteria. This is typical in solicitation-driven selection were the selected providers in one phase become solicited providers in the following phase.
- ***Finding candidates***: The selection pattern does not address how the participant finds the candidates from which to select. Often times the participants will be located via one or more of the following ways:
    - Found via a lookup in a public registry.
    - Retrieved from a "preferred vendor" list or a "trusted partner" list.
    - Tapping registered members of an e-business community or a trading network.

Consequences

- A participant is able to select objectively among similar offerings.
- A participant can optimize some desired quality of the interaction by varying the selection criteria to match some requirements.
- The choice of participant may change from one interaction to the next if the offerings and/or the selection criteria change.
- Keeping a history of the interactions with previously selected partners informs and improves future selection process.

Examples

The MP keeps a list of vendors from which supplies are purchased. Each vendor periodically updates the published price list, minimum order quantities, and the offered service quality such as delivery time. The MP also keeps a record of previous deliveries in terms of how timely they were and the quality of delivered items. When it is time to order new supplies, suppliers that currently have "reasonable" prices and had provided reliable deliveries in the recent past are selected.

Related Patterns

- The ***Selection*** pattern is typically, but not necessarily, associated with a ***Solicitation*** pattern. Participants are solicited for their "offer" then the selection process selects among the submitted offers.
- In the catalogue of workflow patterns [14] "Multiple Choice" patterns and "N-out-of-M" represent possible workflow implementations of the Selection pattern.

5.4. Solicitation Pattern

Context

Some essential information is needed by a party to make a decision that will affect the flow of some interaction that is yet to start. In particular, information about the characteristics of other participants' service offerings is critical to making a decision as to which participant is to be selected for the interaction. The information may not be immediately or publicly available and it can reside completely within the other participant's domains.

Problem

How can the information about the other participants be made available so that the decision can be made in a timely manner?

Forces

- The service offering of the candidate participants is essential information without which the participant wishing to make a decision can not progress.
- The service offering of each participant may change from one point in time to another and from one interaction to the next depending on the specifics of each interaction.
- Information about offerings from solicited participants may not be immediately available to the participant that needs to make a decision.
- The solicitor may need to take the action by some specified time in the future

Solution

The candidate participants are solicited to provide information about their offerings. The soliciting party defines a set of criteria with respect to which the offerings shall be assessed. The solicitor also specifies a deadline for submitting the offerings.

Structure

Figure 5 depicts the structure of the Solicitation pattern. The soliciting participant may involve one or more participants in the solicitation to get the service offering of each. The soliciting party also specifies the criterion of acceptance/assessment of offering and a deadline for submission.

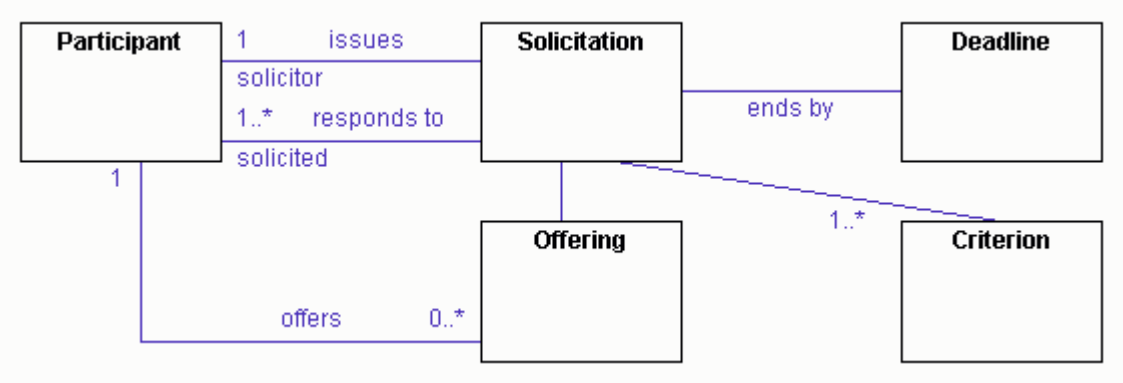

Figure 5. Class diagram showing the Solicitation pattern structure

Considerations and Variants

- ***Multiple solicited participants***: The solicitor typically solicits more than one potential participant for their offerings. A selection will be subsequently made from among the submitted offerings. To optimize the selection process, the solicitor may include information about the selection criteria in the solicitation so that the solicited participants can customize their offerings. The solicitation should also include any information that may be needed by the candidates to propose their offerings.
- ***Public or private***: The solicitor typically informs a selected set of candidates directly (e.g. by sending them a message) about the solicitation and provides enough information for them to present their offerings. In certain situations the candidates are not pre-determined and the solicitation is available to the public. For example, it could be more effective for a huge corporation that has thousands of suppliers to expose a service that allows suppliers to check for upcoming solicitations.
- ***Interaction-specific information***: The solicited information may be specific to the context and content of the interaction and needs to be re-solicited for another interaction of the same type. For example, a vendor solicited for pricing may provide special discounts for large orders.
- ***Adapting to solicited participants interfaces***: The solicitor will need to comply with each solicited participant service interfaces to send the solicitation to each of them. This will be painful unless all the participants comply with some standard interface as in the case of a trading network. This adaptation is not an issue if the solicitor merely exposes his own service that allows interested participants to check for solicitations.
- ***Asynchrony***: Response to the solicitation is typically not received immediately. Creating an offering may require customization for the particular solicitor and the specifics of the upcoming interaction which may require some human decision-making element. In this case, the solicitor will have to specify a callback interface for solicited participants to submit their offerings.

Consequences

- A participant looking for service offerings can find offerings that it needs to progress.
- A participant wishing to make a decision is able to base the decision on up-to-date information.
- The solicitor is able to proceed in a timely manner while still giving solicited participants some time to “prepare” their offerings.

Examples

Before the MP purchases new supplies the available price lists from vendors are consulted. If some of the price lists have expired the vendors are solicited to provide their updated lists. Additionally, the MP provides a service where upcoming requisitions are published so that vendors that are not registered

with the MP may submit their offerings. Each requisition specifies the items to be purchased, the desired quantities, quality specifications for each item, as well as the date the requisition closes.

Related Patterns

- If multiple offerings are solicited a ***Selection*** process usually follows the ***Solicitation***.
- A ***Deadline*** is usually set after which no more offers are accepted for consideration.
- A ***Token*** may be required for the solicited party to submit an offering.
- The "One-to-many send" pattern in the service interaction patterns catalogue [4] is a possible implementation of a ***Solicitation***.

5.5. Deadline Pattern

Context

Service-oriented collaborations involve long-running interactions where asynchronous information exchange takes place between participants. Hours or even days may separate a request for information from the response that provides that information. The infrastructure that relays the messages exchanged between participants will not always be reliable and there could be no direct way of telling whether an expected reply has never been sent or was sent but was lost on the way over.

Problem

How does a party progress in a controlled timely manner when another participant will be providing information asynchronously?

Forces

- The party requiring information cannot wait forever for the other participant to provide the required information.
- There is no guarantee that the required information will be available at a specific time.
- The communication medium may be an unreliable network that does not support "guaranteed message delivery".

Solution

The party requiring the information sets a deadline after which he no longer waits for the required information and an alternate course of action is taken.

Structure

Figure 6 depicts the structure of the ***Deadline*** pattern. A participant specifies a deadline after which if the information required is not available an alternate action shall be taken.

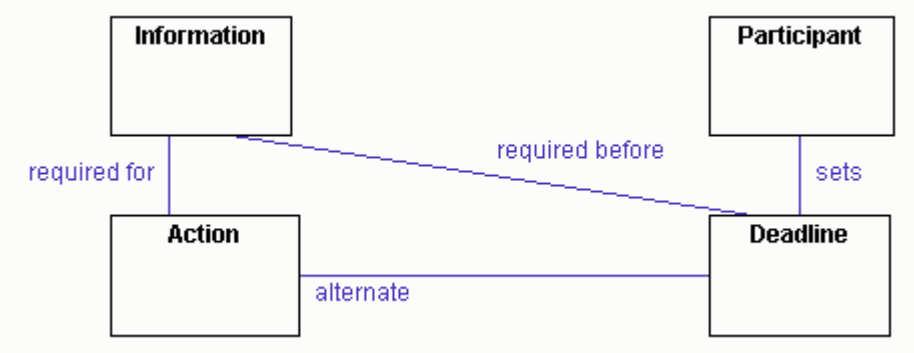

Figure 6. Class diagram showing the Deadline pattern structure

Considerations and Variants

- ***Retry***: A common action to take when a deadline is reached is to retry requesting the required information again. The assumption in this case that something went wrong with the transmission and another attempt to get the information may succeed. Usually the requestor

retries for some maximum number of times before giving up. A retry will be tricky if the request has side effects. In other words, retries are only straightforward if the request is idempotent.

- ***Wait anyway***: If the required information is received before the deadline is reached then the party requiring the information will typically move forward. However, in some cases the party requiring the information will wait till the deadline is reached anyway. A typical example of this is where a ***Solicitation*** has been published and made available to an unknown number of participants then the soliciting party will wait till the deadline before concluding that no more participants will submit an offering.
- ***Absolute or Relative***: The deadline may be specified as an absolute time in the future or relative to some event. For instance, when a solicitation is sent to multiple participants at slightly different times the requesting party may give each solicited participant a number of days to respond from the time they received the solicitation.
- ***Postponement***: In some cases a participant that is not able to fulfill all the requested information before the deadline may submit partial information or no information at all but request an extension to the deadline.
- ***Expiration***: Closely related to a deadline is the concept of expiration. A party that provides information to another participant may attach an expiration date to the information after which the information is deemed to be invalid. A typical example is where a party specifies that an offer is not valid after a certain date.

Consequences

- The party requiring information has some control over the progress of the interaction and does not have to wait forever for the information to become available.
- The interaction becomes tolerant to unreliable communication media.

Examples

When the MP solicits offers from vendors it specifies a date after which no more offers are accepted. The MP waits till the deadline is reached before starting a selection between the vendors who have submitted offers.

An example of Expiration with a relative deadline: When the insurance company issues a treatment authorization it specifies that the authorization has to be used within two weeks from issuance after which it will become void.

Related Patterns

- A ***Solicitation*** is usually associated with a ***Deadline*** for submitting offers.
- A ***Token*** may have an expiration date.
- The concept of a timeout in the constructs of many languages is closely related. An example is the timeout that can be specified when waiting for notification on a monitor in the Java language.

6. Revisiting the Example

To demonstrate and validate the patterns and the relations between them we will apply them to part of the MP example:

We start by the requirements fragment involving treatment authorization (TA). By realizing that the TA is a form of permission that the insurance company gives the MP we can apply the ***Token*** pattern to the requirements snippet resulting in the following questions:

- *Identities*: What identifies each TA? Does each TA have a globally unique ID? Or is the ID unique within the MP/insurance company? What instance of “treat patient” action does the

token enable? Does it enable the "treat patient" action for a certain patient from a specific ailment at a certain date by a certain MP?

- *Multiple-Usage*: Is the MP allowed to reuse the same TA to treat the same patient more than once from the same ailment?
- *Action consuming the token*: When is the TA considered "consumed"? Does the doctor treating the patient submit some report indicating the treatment, the patient, as well as the TA number?

Having applied the Token pattern we consult the diagram of relations between the patterns for what pattern can be potentially applied next which yields both the ***Deadline*** (Expiration) and the ***Correspondence*** patterns. Applying the ***Deadline*** pattern we get to ask:

- *Expiration*: Does the TA ever expire?
- *Absolute or Relative*: How long after issuance does the TA expire?
- *Postponement*: Is the MP allowed to postpone the TA?

Applying the ***Correspondence*** pattern we get to ask:

- *States and Transitions*: What transitions happen to the state of the TA at the insurance company that are of interest to the MP? For instance, can the insurance company cancel the TA after it has issued it to the MP?
- *Clock*: How long after the TA is cancelled does the MP get know about the cancellation?
- *Out-of-date view*: What should happen if the MP gets to know about the cancellation of the TA after it has been used to prescribe a treatment for a patient?
- *Chaining*: Any other correspondence between the state of the TA at the MP and some other participant? For instance, if based on the TA specimens are taken from the patient and sent to an external lab, should the state of the lab tests be affected by the cancellation of the TA? (for instance, does the lab test get cancelled if it had already started, etc.)

Having applied the ***Correspondence*** pattern we again consult the diagram of relations between patterns to find that we can potentially apply the ***Notification*** pattern, and so on.

We now tackle the requirements fragments concerned with purchasing supplies. Realizing that the MP selects among multiple vendors when purchasing supplies we can apply the ***Selection*** pattern to yield these questions:

- *Criteria:* What are the criteria for selecting among vendors? (Pricing, reliability in the past, payment terms, the time it takes to deliver, etc.)
- *Weights on Criteria:* What is the weight on each criterion? Does a vendor who delivers merchandise of variable quality get selected if he offers a considerably lower pricing?
- *Select more than one*: Can a single requisition order be filled from multiple vendors? Do some vendors allow for "tentative orders" (so that they can be selected as "backup")?
- *Finding candidates*: What public listings for vendors are available to the MP? Does the MP keep a list of vendors dealt with in the past?

Having applied the ***Selection*** pattern we refer to the diagram of relations between patterns to find that a ***Selection*** may require a ***Solicitation***. Applying the ***Solicitation*** pattern yields the questions:

- *Multiple solicited participants*: Does the MP solicit multiple vendors? (Obviously yes, as per applying the ***Selection*** pattern). What information does the solicitation include?
- *Public or private*: What is the means by which the MP solicits the vendors? Does the MP make the solicitation publicly available?
- *Interaction-specific information*: Do vendors provide quantity discounts? Do delivery terms differ depending on the requisition?

At this point we can also apply the ***Deadline*** pattern associated with the solicitation.

As can be seen, the application of the approach has yielded a useful set of questions even for such small exemplar fragments.

7. Conclusions and Future Work

In this paper we have attempted to capture a set of commonly occurring patterns in service-oriented interactions involving exchange of information between multiple participants. The focus of the patterns is mainly the requirements of the information exchanged in the interaction rather than the messaging sequence implementing these requirements. The ultimate goal is to elicit and specify the requirements on the interaction in a messaging-sequence-agnostic manner and defer the choice of implementation thereby increasing the flexibility of business "process" description.

Beyond mining for more patterns (e.g. to cover information security aspects), the patterns and their interrelations need much refinement and structuring.

We would like to refine the catalogue to separate patterns that are solely concerned with the requirements of the interaction in terms of *what* information is required and *why* such information is needed. These patterns can then be layered on top of another set of design patterns whose concern is *how* the information is exchanged. As an example, this concept is manifested in the relation between the ***Correspondence*** pattern and the ***Notification*** pattern. This layering will provide guidance on how to proceed from the requirements of SOC interaction to on implementation.

A highly desirable goal is to develop a mechanism for composing patterns into larger patterns that may have more specific semantics. Such patterns will help in composing requirements and asking richer questions. For example, a ***Negotiation*** pattern composes multiple ***Solicitation*** and ***Selection*** instance, which can be further composed with an ***Intermediary*** pattern to yield a ***Brokerage*** pattern. Moreover, we would like to investigate how pattern layering and pattern composition can be combined in one coherent pattern language.

Finally, it is yet to be determined if guidance on applying the patterns can be provided. For instance, a few guidelines on how to match certain bits of requirements to the patterns should make the process of applying the patterns more effective. When the set of patterns and the relations between them become mature it may then be possible to provide some criteria to judge whether all possible steps of applying patterns to a given set of requirements have been taken and all the relevant questions have been asked.

References

1. Andrews, T., et al., *Business Process Execution Language for Web Services Version 1.1*. 2003.
2. Charfi, A. and M. Mezini. *Hybrid Web Service Composition: Business Processes Meet Business Rules*. in *the 2nd international conference on Service oriented computing*. 2004. New York, NY, USA.
3. Christensen, E., et al., *Web Services Description Language (WSDL) 1.1*. 2001, W3C.
4. Barros, A. and M. Dumas, *Service Interaction Patterns: Towards a Reference Framework for Service-Based Business Process Interconnection*. 2005, Faculty of IT, Queensland University of Technology.
5. Hohpe, G. and B. Woolf, *Enterprise Integration Patterns*. 2004: Addison-Wesley.
6. Dwyer, M.B., G.S. Avrunin, and J.C. Corbett. *Patterns in Property Specifications for Finite-state Verification.* in *International Conference on Software Engineering (ICSE)*. 1999. Los Angeles, CA, USA.
7. Li, Z., J. Han, and Y. Jin. *Pattern-Based Specification and Validation of Web Services Interaction Properties*. in *International Conference on Service-Oriented Computing (ICSOC 2005)*. 2005. Amsterdam, The Netherlands: Springer.
8. *The Patterns Homepage*.

9. Mulyar, N.A. and W.M.P.v.d. Aalst. *Patterns in Colored Petri Nets.* in *Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*. 2005. Aarhus C, Denmark.
10. Fernandez, E.B. and R. Pan. *A Pattern Language for security models.* in *PLoP 2001*. 2001.
11. Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994: Addison-Wesley.
12. Jackson, M., *Problem Frames - Analyzing and structuring software development problems*. 2001: Addison-Wesley.
13. Kavantzas, N., et al., *Web Services Choreography Description Language Version 1.0*. 2005, W3C.
14. Aalst, W.M.P.v.d., et al., *Workflow Patterns.* Distributed and Parallel Databases, 2003. **14**(3): p. 5–51.