Strict and Extended Interpretations of Operation Contracts in Conceptual Modeling

Anna Queralt and Ernest Teniente

Universitat Politècnica de Catalunya
Dept. de Llenguatges i Sistemes Informàtics
c/ Jordi Girona 1-3, 08034 Barcelona (Catalonia, Spain)
{aqueralt, teniente}@lsi.upc.edu

Abstract. This paper describes two different ways of understanding operation contracts in conceptual modeling. The main difference between them lies in the way operation postconditions and integrity constraints are guaranteed, which impacts on the desirable properties of operation contracts according to recommended good practices for requirements specification. Both approaches are formalized and then compared in a number of issues.

1. Introduction

An information system maintains a representation of the state of a domain in its information base. The state of the information base is the set of instances of the entity and relationship types defined in the conceptual schema. Additionally, a conceptual schema contains a set of integrity constraints that define conditions that each state of the information base should satisfy.

The content of the information base changes due to the execution of operations. The effect of an operation on the information base is usually defined by means of pre and postconditions. A precondition expresses a condition that the information base must satisfy when the call to the operation is done. A postcondition expresses a condition that the information base must satisfy after the application of the operation.

Several books on conceptual modeling deal both with the structural and the behavioural part of a conceptual schema and give precise definitions for integrity constraints and also for pre and postconditions [5, 10, 11, 13, 17, 19-21]. However, they usually pay little attention to the precise semantics of operation contracts since in general they do not establish an explicit relation between operation postconditions and the integrity constraints defined in the conceptual schema.

This is an important open issue in conceptual modeling since the semantics of an operation should precisely define both the conditions under which the operation can be applied as well as the new information base state obtained as a result of its execution. The latter cannot be done if we do not establish precisely how the satisfaction of integrity constraints is guaranteed once the operation postcondition is satisfied.

In this paper we propose two different interpretations to define the semantics of operation contracts: a *strict* interpretation and an *extended* one. The main difference between them lies in the way postconditions and integrity constraints are guaranteed.

A strict interpretation assumes a passive behaviour of operations, since it prevents an operation from being applied if some integrity constraint is violated (although both its pre and postconditions are satisfied). Instead, an extended interpretation entails a reactive behaviour of operations, since it must take care of maintaining integrity constraints satisfaction whenever they are violated, so that the operation will always be applied if its precondition is satisfied.

We provide formal definitions for both strict and extended interpretations and we illustrate them with several examples. Moreover, we also compare them in a number of issues from the point of view of the characteristics of a good software specification.

The main contribution of our work is twofold. First, we clearly state the relationship between integrity constraints and operation postconditions, an issue not yet enough clarified in conceptual modeling. Second, we show that a strict interpretation of operation contracts provides several advantages over an extended one as far conceptual modeling is concerned.

Our work is independent of any specific conceptual modeling language and, thus, our results can be applied to any of them provided that it allows the definition of operations and explicit integrity constraints. In addition, our results can be applied to almost all current conceptual modeling approaches like [5, 10, 11, 13, 15, 17, 19].

Next section reviews related work. Section 3 presents basic concepts and an example used throughout the paper. Section 4 defines the strict and extended interpretations, while section 5 compares them. In section 6 we provide some experimental results and, finally, we present our conclusions and point out future work in section 7.

2. Related Work

Previous work on conceptual modeling provides precise definitions for integrity constraints and also definitions for pre and postconditions, but sometimes without explicitly establishing a clear relation between them. In this way, they generally pay little attention to the precise semantics of operation contracts, in the sense of how the satisfaction of integrity constraints is guaranteed after each operation execution.

In many proposals, integrity constraints are usually kept separated from the discussion about operation contracts. In [11, 17, 19-21] definitions are given for integrity constraints and for pre and postconditions, but the implications that constraints have on the way of specifying operation contracts are not discussed. Even in [10] it is said explicitly that integrity constraints are not included in the discussion about pre and postconditions for the sake of simplicity.

On the contrary, the relationship between operation contracts and integrity constraints is clearly established in [5, 13, 15]. According to them, the state of the information base resulting from the execution of an operation must satisfy both the postcondition and the integrity constraints every time the precondition is satisfied. As we will see, this semantics corresponds to our extended interpretation of operation contracts, which as we will discuss presents several drawbacks from the point of view of the characteristics of a good software specification.

The previous discussion does not concern proposals that do not allow the explicit definition of integrity constraints in the conceptual schema, like Taxis [14].

3. Basic Concepts

The conceptual schema of an information system must include all relevant static and dynamic aspects of its domain [9]. The part of a conceptual schema that deals with static aspects is called the structural schema, and the part that deals with dynamic aspects is called the behavioural schema.

A structural schema consists of a taxonomy of entity types together with their attributes, a taxonomy of relationship types among entity types, and a set of integrity constraints over the state of the domain, which define conditions that each state of the information base must satisfy. Those constraints may have a graphical representation or can be defined by means of a particular language.

The content of the information base changes due to the execution of operations. A behavioural schema contains a set of operations and the definition of their effect on the information base. This knowledge is usually defined by the preconditions and postconditions of the operations. A precondition expresses a condition that must be satisfied when the call to the operation is done. A postcondition expresses a condition that the new state of the information base must satisfy. Changes in an information base state are defined by means of a set of one or more structural events to be applied, which are drawn from the operation pre and postconditions.

A structural event is an elementary change in the population of an entity or relationship type, i.e. the creation or deletion of instances. The precise number and meaning of structural events depend on the conceptual modeling language used. In this paper, we will use the following kinds of structural events[1]:

- Entity insertion: *insert(EntityType(attribute$_1$,...,attribute$_n$))*.
- Entity deletion: *delete(EntityType(object))*.
- Entity generalization (an object is moved from an entity type to its supertype): *generalize(object, EntityType)*.
- Entity specialization (an object is moved from an entity type to one of its subtypes): *specialize(object, EntityType(attribute$_1$,...,attribute$_n$))*.
- Relationship insertion: *newLink(RelationshipType(participant$_1$,...,participant$_n$, [attr$_1$,...,attr$_n$]))*.

The application of a set of structural events to a state *IB* of the information base results in a new state *IB'* of the same information base. Given a state of the information base *IB*, there are several sets of structural events that lead to new states satisfying an operation postcondition. From those, we are only interested in the minimal ones. A set *S* of structural events is minimal if no proper subset of *S* is also a set of structural events that satisfies the postcondition. This is the way in which we deal with the frame problem [2].

We present a conceptual schema that models the domain of Internet auctions, which will serve as an example throughout the paper. In an auction site, items are owned by users, and may be auctioned for a certain period of time, during which the auction is open. Users registered in the site can place several bids for an auctioned item, as long as the auction is not closed. The amount of a new bid must be greater than all the bids placed so far.

[1] Other kinds of structural events exist but we only define those used in our examples.

Figure 1 shows a UML class diagram specifying this information. The additional constraints needed are textually defined and then formalized in OCL [16]. Without loss of generality, we will also make a textual description and an OCL formalization of operation contracts.

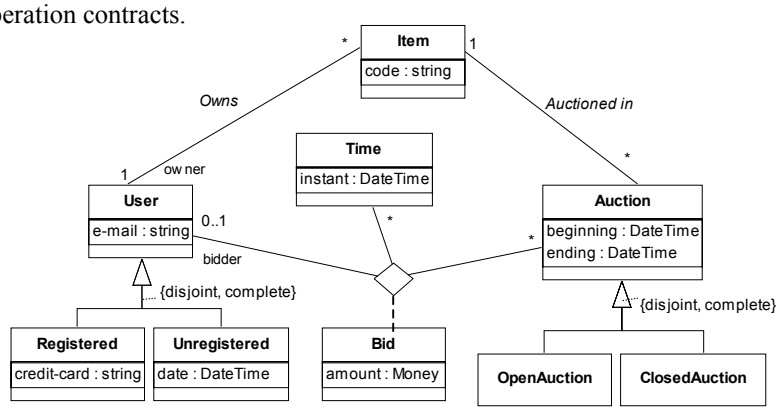

Fig. 1. Class diagram for Internet auctions

Integrity constraints

- Users are identified by e-mail

```
context User inv emailIsUnique:
User.allInstances()->isUnique(e-mail)
```

- Items are identified by code

```
context Item inv codeIsUnique:
Item.allInstances()->isUnique(code)
```

- The amount of a bid must be greater than the amount of all the previous bids of the same auction

```
context Bid inv amountAbovePreviousBids:
self.auction.bid->select(time.instant < self.time.instant)->
forall(amount < self.amount)
```

4. Semantics of Operation Contracts

As we said, an operation precondition expresses requirements that any call must satisfy in order to be correct, while its postcondition expresses properties that are ensured in return by the execution of the call [12].

In addition to preconditions and postconditions, invariants play an important role in the definition of the semantics of operation contracts. Every operation may assume that the invariants are true when it is entered and must in return ensure that they are true on its completion [8]. From the point of view of conceptual modeling, invariants capture semantic properties of the domain and are represented by means of integrity constraints that restrict the possible states of the information base. In this sense, those constraints express global properties of the state and, thus, they must be preserved by all the operations. The semantics of operation contracts in conceptual modeling we identify in this paper differ mainly on the way integrity constraints are considered.

Given an information base *IB* and an operation *Op*, the semantics of *Op* defines the conditions under which *Op* can be applied and the new *IB'* we obtain as a result of applying *Op* to *IB*.

4.1. Strict and Extended Interpretations of an Operation Contract

Let *IB* and *IB'* be states of the information base. Let *Op(IB, IB')* denote that *IB'* is the result of applying an operation *Op* on *IB*. Let *Pre* and *Post* be the precondition and the postcondition of *Op* and *S* be the minimal set of structural events that satisfies *Post* when applied to *IB*[2]. Let *IB' = S(IB)* denote that *IB'* is obtained as a result of applying to *IB* all the structural events in *S*. Let *IC* be the set of integrity constraints defined in the conceptual schema.

Definition 1: *Strict Interpretation of an Operation Op*

$\forall$ *IB*, *IB'* such that *Op(IB, IB')* the following three conditions hold:

a) $IB \models Pre \wedge IC$
b) $IB' = S(IB)$
c) $IB' \models Post \wedge IC$

Intuitively, the first condition states that there is a transition from an information base *IB* to an information base *IB'* as a result of applying an operation *Op* only if *IB* satisfies *Pre* and it is consistent (i.e. it satisfies all integrity constraints). Moreover, according to the second condition, *IB'* is obtained exactly as a result of applying to *IB* the minimal set *S* of structural events that satisfies *Post*. Finally, the third condition requires *IB'* to satisfy *Post* (always true according to b)) and to be consistent. If any of the conditions does not hold, then *Op* is not applied to *IB*.

To illustrate the previous definition, assume the following contract for an operation *newUser* aimed at registering new users in our Internet auctions domain:

```
Operation: newUser(email: String, creditCard: String)
Pre:       --there is not a user with e-mail=email
           not User.allInstances() → exists(u|u.e-mail=email)
Post:      --a new instance u of Registered, identified by
           email and with credit-card=creditCard is created
           u.oclIsNew() and u.oclIsTypeOf(Registered) and
           u.e-mail = email and u.credit-card = creditCard
```

Fig. 2. Contract for the operation *newUser*

We have that *S*={*insert(Registered(email,creditCard))*} suffices to satisfy the post-condition of *newUser(email,creditCard)* in any *IB*.

We are going to show the conditions under which *newUser* can be applied to *IB* and the new *IB'* resulting of applying *newUser* according to the strict semantics. We distinguish two different situations depending on the contents of *IB*:

1. *IB* contains a user identified by *email*. In this case, the operation may not be applied since condition a) of the strict interpretation is not satisfied.

[2] Exceptionally, several minimal sets S may exist but only in those cases when some kind of random behaviour is desired. Any of them can be arbitrarily chosen then.

2. *IB* does not contain any user identified by *email*. Then:
   - a) is guaranteed, since *Pre* is satisfied and *IB* is consistent.
   - b) states that *IB' = S(IB)*, i.e. *IB'* is obtained just by inserting a registered user with *email* and *creditCard* into *IB*.
   - *IB'*, as obtained according to b), satisfies c) since *S* necessarily satisfies the postcondition and applying *S* to *IB* never violates any integrity constraint. Note that the only integrity constraint that *newUser* may violate is the first one (*emailIsUnique*) but this will never happen when *IB* does not contain any user with the same *email*.

Summarizing, we have that according to a strict interpretation the semantics of the operation *newUser* is the following. If *IB* does not contain any user identified by *email* then the operation results on the application of the structural event *insert(Registered(email,creditCard))*. Otherwise, the operation may not be applied since *Pre* is not satisfied and, thus, not all the necessary conditions hold.

Definition 2: *Extended Interpretation of an Operation Op*

$\forall$ *IB*, *IB'* such that *Op(IB, IB')* the following four conditions hold:

a) $IB \models Pre \wedge IC$

b) $IB' = S_2(IB)$ and $S \subseteq S_2$

c) $IB' \models Post \wedge IC$

d) $\neg\exists\ S_3, S \subseteq S_3 \subset S_2$ such that $S_3(IB) \models Post \wedge IC$

The first condition states, as before, that the transition is only possible if *IB* satisfies *Pre* and it is consistent. The second condition asserts now that to obtain *IB'* at least the structural events in *S* must be applied. However, it does not discard the application of additional structural events to *IB*. The third condition requires *IB'* to satisfy *Post* and to be consistent. Finally, the fourth condition states a minimality condition on S_2 in the sense that there is no proper subset of S_2 that satisfies *Post* and *IC*.

From previous proposals, [5, 13, 15] follow an extended interpretation to define the semantics of operation contracts.

As we said before, the main difference between both interpretations relies on how integrity constraints are handled. As stated in Definition 1, a strict interpretation defines a set of structural events *S,* determined only by the postcondition, to perform the transition from *IB* to *IB'* according to *Op*. *S* may only be applied to *IB* if it does not lead to a violation of any integrity constraint. Otherwise, *Op* must be rejected.

On the contrary, an extended interpretation allows for several different sets of structural events S_i to be applied to *IB*, provided that all of them include at least the events in *S* and satisfy the minimality condition. The additional structural events in S_i must be such that they guarantee that no constraint is violated in the resulting state, even though some of them were violated by the events in *S*. Clearly, if *S* itself does not violate any constraint there is no need to consider additional structural events.

If we assume now an extended interpretation of the operation in Figure 2, we have that when *IB* contains a user identified by *email* the operation will not be executed. If this does not happen, the application of *S*={*insert(Registered(email,creditCard))*} suffices also to satisfy the four conditions of Definition 2, following the same reasoning as for the strict interpretation. Hence, we have in this particular example that the se-

mantics of strict and extended interpretations coincide. The main reason for that coincidence is that no integrity constraint is violated by *S* and, thus, the application of *S* is enough to obtain *IB'* in both cases.

We see, however, an important drawback in the previous operation contract. The problem is that, in this example, the operation precondition is redundant since the same aspect of the specified system (users are identified by e-mail) is already guaranteed by the first integrity constraint. As stated in [3], non-redundant conceptual schemas provide several advantages regarding desirable properties of software specifications and ease of design and implementation.

To avoid redundancy in the specification of the operation *newUser* we should define an operation contract like the one in Figure 2 but with an empty precondition. The set *S* will be the same (*S*={*insert(Registered(email,creditCard))*}). However, we have now that the semantics of both interpretations does not coincide anymore.

In a strict interpretation, the semantics of *newUser* would be the same as the one stated above. We show it by distinguishing again the same situations:

1. *IB* contains a user identified by *email*. Then:
   - a) is guaranteed, since *Pre* is True and *IB* is consistent.
   - b) states that *IB' = S(IB)*, i.e. *IB'* is obtained just by inserting a registered user with *email* and *creditCard* into *IB*.
   - c) is not satisfied since *IB'* will always violate the first integrity constraint.

   We have then that *newUser* may not be applied in such an *IB* since it is impossible to satisfy all conditions required by a strict interpretation.
2. *IB* does not contain any user identified by *email*. Then:
   - a) is guaranteed.
   - b) states that *IB' = S(IB)*.
   - *IB'*, as obtained according to b), satisfies c) since *S* necessarily satisfies the postcondition and applying *S* to *IB* never violates any integrity constraint.

In an extended interpretation, we must distinguish also the same two situations:

1. *IB* contains a user *user* identified by email. Then:
   - a) is guaranteed.
   - b) states that $IB' = S_2(IB)$ and $S \subseteq S_2$. i.e., any information base resulting from applying a set S_2 of structural events that contains *S*={*insert(Registered(email,creditCard))*} will satisfy this condition.
   - c) states that *IB'* must imply both the postcondition (this is always true since *S* necessarily satisfies it) and all integrity constraints. Since *S* itself violates the first integrity constraint, S_2 must be a superset of *S*.
   - d) states that S_2 must be minimal, and adding *delete(User(user))* to *S* suffices both to satisfy this condition and repair the previous violation.
2. *IB* does not contain any user identified by email. The behaviour in this situation is the same as the one of the strict interpretation.

Summarizing, the semantics of the non-redundant operation contract for *newUser* according to an extended interpretation is the following. If *IB* does not contain a user identified by *email* the operation results in the application of the structural event *insert(Registered(email,creditCard))*. Otherwise, i.e. *IB* contains a user *u* identified by

email, it results on the application of $S_2 = S \cup \{delete(User(user))\}$, in order to satisfy the postcondition and repair the violation.

As a result of the previous discussion, we may conclude that an extended interpretation implicitly entails a reactive behaviour of operations, since it must take care of maintaining integrity constraints satisfaction whenever they are violated, so that the operation is always be executed if its precondition is satisfied (and it is possible to maintain all integrity constraints). Instead, a strict interpretation assumes a passive behaviour of the operations, since it prevents the operation from being applied if some integrity constraint is violated.

We may establish a certain parallelism between the semantics we have just defined and the notions of partial and total correctness in axiomatic computer programming [7]. In fact, our strict interpretation semantics is somehow inspired by partial correctness, when adapted to conceptual modeling and taking integrity constraints into account. In a similar way, our extended interpretation is inspired by total correctness. The similarity between our semantics and those of axiomatic programming lies in the fact that in total correctness the postcondition is always satisfied as long as the precondition is, as in our extended interpretation. On the other hand, in partial correctness, the satisfaction of the precondition does not necessarily imply the satisfaction of the postcondition, as happens in our strict interpretation of operation contracts.

4.2. More on the Strict Interpretation

We will further illustrate the semantics of the strict interpretation by means of some examples. We start by specifying the following contract for an operation *bid*, aimed at allowing users to place bids in our Internet auctions domain:

```
Operation:  bid(u:Registered, a:OpenAuction, amt:Money)
Pre:
Post:       --a new instance b of Bid, with amount amt and de-
            fined by the user u, the auction a and the current
            time is created
            b.oclIsNew() and b.oclIsTypeOf(Bid) and b.bidder = u
            and b.auction=a and b.amount=amt and b.time=now()
```

Fig. 3. Contract for the operation *bid*

For the sake of non-redundancy, we assume that the precondition is empty, but the behaviour of this operation would be exactly the same if it included (redundantly) statements to guarantee that no integrity constraint is violated.

The minimal set of structural events that satisfies *Post* is $S=\{newLink(Bid(u,a,currentTime,amt))\}$. So, according to a strict interpretation, the semantics of *bid* is the following. If *amt* is greater than the amount of all the previous bids of *a*, then the operation results on the application of the structural event *newLink(Bid(u,a,currentTime,amt))*. Otherwise, the operation cannot be applied since it would violate the constraint *amountAbovePreviousBids*. Note that this interpretation corresponds with the expected semantics of an operation *bid*.

The next example is useful to illustrate the main drawback of the strict interpretation. If we want to define an operation *unregisterUser* aimed at unregistering registered users, we could specify the following operation contract:

```
Operation:  unregisterUser(u:Registered)
Pre:
Post:       --user u becomes instance of Unregistered in the
            current time and ceases to be instance of Registered
            u.oclIsTypeOf(Unregistered) and
            u.oclAsType(Unregistered).date = now() and
            not u.oclIsTypeOf(Registered)
```

Fig. 4. Contract for the operation *unregisterUser*

According to a strict interpretation, the minimal set of structural events that satisfies *Post* is S={*specialize(u, Unregistered(now)), generalize(u, Registered)*}. Then, according to Definition 1, *IB'* is obtained from *IB* by inserting *u* as an instance of *Unregistered* and removing him from *Registered*. Note that this operation always results in a state *IB'* satisfying both the postcondition and the integrity constraints.

Clearly, the previous semantics is the expected one for the operation *unregisterUser*. However, the postcondition of the contract explicitly states that *u* must not be a registered user any more. Moreover, the class diagram already entails that if *u* is *Unregistered* (as enforced also by the contract postcondition) *u* may not be *Registered* (because of the disjointness constraint). So, we could argue whether the same behaviour could be achieved by removing *"not u.oclIsTypeOf(Registered)"* from *Post*.

The problem is that if we do that, a strict interpretation of the contract would never allow to apply the operation *unregisterUser* since the information base resulting from just inserting *u* as an unregistered user (the minimal set of structural events that would satisfy now the postcondition) would always violate the disjointness constraint. Hence, a strict interpretation requires to state *"not u.oclIsTypeOf(Registered)"* in the operation postcondition.

4.3. More on the Extended Interpretation

We will further illustrate first the semantics of the extended interpretation by taking the previous example into account and assuming the following contract for the operation *unregisterUser*:

```
Operation:  unregisterUser(u:Registered)
Pre:
Post:       --user u becomes now instance of Unregistered
            u.oclIsTypeOf(Unregistered) and
            u.oclAsType(Unregistered).date = now()
```

Fig. 5. Contract for the operation *unregisterUser*

According to Definition 2, the semantics of the previous contract when assuming an extended interpretation is the following:

- a) is guaranteed.
- b) states that $IB' = S_2(IB)$ and $S \subseteq S_2$, S={*specialize(u, Unregistered(now))*}.
- Note that the application of S alone would violate the disjointness constraint of the *User* specialization. Then, according to c), S_2 must be a superset of S.
- The minimal superset of S that satisfies both *Post* and *IC* is $S_2 = S \cup$ {*generalize(u, Registered)*}.

As a conclusion, we have in this example that an extended interpretation does not present the same drawback than a strict one, since we do not need to specify in the postcondition of *unregisterUser* that *u* is not registered anymore. The reason is that the reactive behaviour of an extended interpretation when some constraint is violated will suffice to detect it (even if it is not explicitly specified, as in the contract of Figure 5) and conclude that the previous contract requires always the application of two structural events: {*specialize(u, Unregistered(now)), generalize(u, Registered)*}.

To illustrate the main drawback of the extended interpretation we will consider again the contract of Figure 3 for an operation *bid*. As before, we assume that the operation has no precondition (to achieve non-redundancy) while the postcondition asserts that a new *bid* is created for the given registered user and open auction. We distinguish two relevant situations:

1. *amt* is greater than the amount of all the previous bids of *a*. Then the operation results on the application of *S*={*newLink(Bid(u,a,now,amt))*}, the minimal set of structural events that satisfies *Post*, since it does not violate any constraint.
2. On the contrary, the application of *S* alone would result in the violation of the integrity constraint *amountAbovePreviousBids*. To avoid this violation, additional structural events should be taken into account. In particular, they should repair the violation either decreasing or eliminating previous bids.

According to this, the previous operation contract admits two different sets of structural events to be applied to *IB*. In addition to the events in *S*, one of them contains structural events to decrease the amount of previous bids, while the other contains the deletion of previous bids (note that both of them satisfy both *Post* and *IC*). Clearly, both alternatives correspond to completely different business rules. Hence, a random behaviour of such operation contract is not acceptable.

This is a clear example to show that assuming an extended interpretation can easily lead to ambiguous contracts. The problem is that, as stated in the *IEEE Recommended Practice for Software Requirements Specifications* (SRS) [1], a good SRS must be unambiguous in the sense that each requirement, and in particular the ones stated in the operation contracts, must have a unique understanding.

There are two possible ways to avoid ambiguity. One possibility is to strengthen the operation precondition to ensure that no integrity constraint violation will be produced. In the previous example this would be done with the contract:

```
Operation:  bid(u:Registered, a:OpenAuction, amt:Money)
Pre:        --the amount amt is above the amount of all previous
            bids for the same auction
            a.bid-> forall(amount < amt)
Post:       --a new instance b of Bid, with amount amt and de-
            fined by the user u, the auction a and the current
            time is created
            b.oclIsNew() and b.oclIsTypeOf(Bid) and b.bidder=u
            and b.auction=a and b.time=now() and b.amount=amt
```

Fig. 6. A redundant contract for the operation *bid*

The problem is that this way to avoid ambiguity results in a redundant operation precondition. As we said before, this situation (although being acceptable) would not follow suggested good practices of conceptual modeling [3].

The second possibility in order to avoid ambiguity is to explicitly state in the operation postcondition the way to repair integrity constraints violations, as we have to do in a strict interpretation. This could be done in the previous example by choosing, for each integrity constraint, one of the two possible ways to repair the constraint violation (i.e. either to decrease the amount of previous bids or to delete them) and specifying the corresponding OCL expression in the operation postcondition.

Note that there are some situations where an extended interpretation is not ambiguous without the need for specifying additional information in the operation contracts. The example of Figure 5 allows illustrating them. In this example, a disjointness constraint is violated and, since the only possible way to repair it is by deleting *u* as a registered user, no ambiguity exists at all. In general, the extended interpretation of an operation contract will not be ambiguous when all integrity constraints that are violated by the operation execution allow only for a single repair.

4.4 Summarizing Strict and Extended Interpretations

As we said, the main differences between strict and extended interpretations lie in the way integrity constraints are enforced. That is, when there is no violation of integrity constraints, the semantics of a given operation contract is equivalent in both interpretations. In this case, the new information base state is always obtained by the application of the minimal set *S* of structural events that satisfy the operation postcondition.

However, if the violation of some constraint occurs when applying the structural events in *S*, the semantics of the contract depends on the interpretation chosen.

On the one hand, under a strict interpretation the violation of a constraint entails that the operation is not applied and, thus, the information base remains unchanged. For this reason, no redundant checks in the precondition are needed. A strict interpretation of the contract of *bid* in Figure 3 serves as an example of this situation.

Alternatively, if we do not want the operation to be rejected, its postcondition must explicitly state how to satisfy the constraints after the operation execution. This can be seen in the contract for *unregisterUser* in Figure 4.

On the other hand, under an extended interpretation the operation may be applied even though some constraint is violated by *S,* and the new state of the information base is obtained by means of the application of a set of structural events *S'*, a superset of *S*, that guarantees that no constraint is violated. Note that *S'* is unique as long as there is a single way to guarantee the constraints, as shown in section 4.3 with a disjointness constraint violation. If *S'* is not unique, as usually happens in practice, the operation contract is ambiguous. To avoid ambiguities, we must specify in the postcondition the way to preserve consistency (as must be done in a strict interpretation).

Alternatively, if we want to maintain consistency by rejecting the operation, its precondition must include redundant checks to ensure that the operation execution does not violate the constraints. An example of this situation can be found in the contract for the operation *bid* specified in Figure 2.

We summarize in Table 1 how (in addition to the intended behaviour of the operation) we should specify the treatment of integrity constraints in a contract, depending both on the interpretation chosen and whether we want the operation to be rejected or applied when the events in *S* violate some integrity constraint.

Table 1. Avoiding integrity constraint violations

	Reject the operation	**Apply the operation**
Strict	Nothing else needs to be done	Specify how to satisfy the constraints in *Post*
Extended	Add redundant checks to *Pre*	Specify how to satisfy the constraints in *Post* (if the contract is ambiguous)

5. Discussion

We compare both approaches from the point of view of the relevant characteristics of a good software requirements specification (SRS) [1, 4]. As recommended also in [1, 4], we assume an unambiguous operation contract specification. In general, this is always true in a strict interpretation, since a deterministic behaviour is usually desired. Regarding the extended interpretation, we will concentrate in this section on those situations where non-ambiguity is achieved by strengthening either the operation pre or postcondition, since this is the most frequent case in practice.

5.1. Completeness

An SRS is complete if it includes, among others, the definition of the responses of the software to all realizable classes of input data in all realizable classes of situations.

From this point of view, both approaches can be considered complete. An extended interpretation avoids erroneous execution of an operation by means of its precondition while a strict one assumes that the response to undesired situations is the rejection of the changes done by the operation. Moreover, when the precondition is not satisfied, both approaches act the same way: rejecting the operation.

For instance, understanding the operation *bid* (see Figure 3) from the point of view of an extended interpretation, we always obtain a defined result when the precondition is satisfied, which is exactly the one specified in the postcondition (a new bid is created and associated to the user and auction specified as parameters and the current time) plus the additional changes, if any, required to satisfy all integrity constraints.

Understanding *bid* from a strict interpretation, we have two kinds of results when the precondition holds. In those cases where no integrity constraint is violated, the resulting state of the information base is the one specified in the postcondition, as occurs with an extended interpretation. On the other hand, when some integrity constraint is violated, *bid* is rejected and the information base remains unchanged.

5.2. Consistency

An SRS is consistent if, and only if, no subset of individual requirements described in it conflict.

Although none of the approaches leads directly to an inconsistent specification, a strict interpretation facilitates having a consistent one while an extended interpretation is more prone to the specification of conflicting requirements. The reason is that,

since integrity constraints are sometimes specified both in the structural schema and as preconditions of operations in the behavioural schema, they can be in contradiction and, therefore, lead to an inconsistent specification.

For instance, it will be more difficult to keep the specification of *bid* consistent with the operation contract specified in Figure 6 than with the one specified in Figure 3, both of them having the same behaviour. The reason is that we could easily have specified in the precondition of the first contract that the amount of a bid must be greater or equal than the amount of the previous bids for the same auction, which would be clearly inconsistent with the integrity constraint *amountAbovePreviousBids,* which forces a bid to be higher than the previous ones.

5.3. Verifiability

An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement.

Unlike the previous criterion, verifiability is more easily achieved by an extended interpretation. Although both approaches allow the verification of the software product, this process can become more complicated assuming a strict interpretation due to the dispersion of the requirements affecting an operation.

For example, to verify the correct behaviour of the operation bid as defined in Figure 3 we must also take into account the integrity constraint *amountAbovePrevious-Bids*. Taking, however, the contract in Figure 6, no additional information is needed in order to verify it.

5.4. Modifiability

An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently. Modifiability generally requires an SRS not to be redundant.

With regards to modifiability, again an extended interpretation is more prone to errors due to the necessary duplication of integrity constraints in the preconditions. When changing a requirement, it is easy to forget changing it in every precondition it appears, which, as well as wasting more time, can lead to inconsistencies.

Suppose that a requirement changes, for instance we want to enforce that a bid may only be placed if it is a 5 % higher than the highest previous one for the same auction. In this case, the integrity constraint *amountAbovePreviousBids* must be changed in order to express this requirement. Moreover, with an extended interpretation we will also have to modify the precondition of the operation *bid* (see Figure 6) stating again the same condition in order to be consistent, and take care of doing the same in every contract affected by the change. However, with a strict interpretation, we need not make any additional changes, since requirements stated by integrity constraints are stated only in the structural schema.

We find, much less frequently, a similar drawback when the postcondition already specifies how to maintain a certain integrity constraint. This drawback is shared by

both approaches. On the one hand, a strict interpretation needs to specify this reactive behaviour always in the postcondition. On the other, an extended interpretation requires to do the same to guarantee there is only one possible reaction to the violation of each integrity constraint.

5.5. Conciseness

Given two SRS for the same system, each exhibiting identical levels of all the previously mentioned qualities, the SRS that is shorter is better.

Taking conciseness into account, it is clear that the a strict interpretation approach helps to get shorter specifications, since each integrity constraint is specified in exactly one place.

It can easily be seen just by comparing again the contracts of Figures 3 and 6. It is clear that the one in Figure 3, corresponding to a strict interpretation, is shorter and, however, both of them have the same meaning.

5.6. Summary

The following table summarizes the discussion in this section. Rows correspond to desirable properties of a good software requirements specification while columns refer to the interpretations we have defined in this paper. A ✓ in a cell denotes the convenience of the corresponding interpretation in order to achieve the property.

Table 2. Comparison of the approaches

	Extended interpretation	**Strict interpretation**
Completeness	✓	✓
Consistency		✓
Verifiability	✓	
Modifiability		✓
Conciseness		✓

As we can see, completeness is achieved in both interpretations; consistency, modifiability and conciseness are easier to achieve in a strict interpretation; and verifiability in an extended one. For this reason, we may conclude that a strict interpretation of operation contracts provides in general several advantages over an extended one regarding conceptual modeling.

6. Experimental Tests

We have applied the ideas discussed in this paper in the specification of some real-life applications, far beyond the simple examples we have used so far to illustrate them. In particular, we have developed a generic conceptual schema for the e-marketplace domain [18]. This conceptual schema is specified in UML, in combination with OCL, to formalize the structural and the behavioural properties of this information system.

Our generic e-marketplace conceptual schema has been drawn from an external study of some well-known e-marketplaces and auction-sites like *eBay, OnSale* and *Amazon*, as well as the job search site *Monster*, and it covers the main functionalities provided by an e-marketplace: determining product offerings, searching for products and price discovery. The whole specification includes about ninety integrity constraints and fifty operations that may modify the state of the information base.

We used a strict interpretation of operation contracts in the specification of this conceptual schema. The main reason for this choice was the requirement to avoid ambiguity. In an extended interpretation, this is achieved by (redundantly) including the information provided by the constraints in the preconditions of the operations that may violate them. However, with such a huge amount of operations and integrity constraints it becomes very difficult to identify all integrity constraints that may be violated by an operation execution.

On the contrary, nothing had to be done to avoid ambiguity in a strict interpretation. Therefore, the assumption of a strict interpretation notably simplified the task of developing the specification. This is an important additional advantage of a strict interpretation beyond the ones already pointed out in the previous section.

Another example that assumes a strict interpretation of operation contracts can be found in [6], where a conceptual schema for EU-Rent, a car rental company which provides typical rental services, is provided. This conceptual schema consists of about fifty integrity constraints and eighty non-query operations, and similar conclusions can be drawn from its specification regarding the different semantics of operation contracts.

7. Conclusions and Further Work

The main goal of this paper has been to clarify the semantics of operation contracts in conceptual modeling. In this sense, we have proposed two different interpretations of operation contracts, a *strict* and an *extended* one, which differ on the way to understand the relationship between operation postconditions and integrity constraints.

Roughly speaking, a strict interpretation prevents an operation from being applied if some integrity constraint is violated. On the contrary, an extended interpretation assumes its semantics must take care of maintaining integrity constraints when they are violated as a consequence of applying the events that satisfy the postcondition.

We have provided formal definitions for the strict and the extended interpretations and we have compared them in a number of issues from the point of view of the characteristics of a good software specification. From our discussion, we may conclude that a strict interpretation does better than an extended one as far as consistency, modifiability and conciseness of the conceptual schema are concerned; an extended is better regarding verifiability, and they both present similar contributions with respect to completeness. Moreover, from our experimental tests we have learnt that a strict interpretation considerably simplifies the task of developing a specification. In this way, we have shown that a strict interpretation of operation contracts provides several advantages over an extended one.

Our work is independent of any specific conceptual modeling language and, thus, our results can be applied to any of them provided that it allows the definition of integrity constraints in the structural schema and operations in the behavioural one.

Further work may be devoted to study whether state diagrams impact in the interpretations we have proposed since there is a close relationship between the events that cause state transitions, operation postconditions and integrity constraints.

References

[1] IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998). (1998)

[2] Borgida, A., Mylopoulos, J., Reiter, R.: On the frame problem in procedure specifications. IEEE Transactions on Software Engineering 21(10) (1995) 785-798

[3] Costal, D., Sancho, M. R., Teniente, E.: Understanding Redundancy in UML Models for Object-Oriented Analysis. In: Advanced Information Systems Engineering: 14th International Conference, CAiSE 2002 Proceedings. LNCS 2348 (2002) 659-674

[4] Davis, A. M.: Software Requirements: Objects, Functions and States. Prentice Hall, Englewood Cliffs (1993)

[5] D'Souza, D. F., Wills, A. C.: Objects, Components and Frameworks. The Catalysis Approach. Addison-Wesley (1998)

[6] Frias, L., Queralt, A., Olivé, A.: EU-Rent Car Rentals Specification. Departament de LSI, UPC, Technical Report LSI-03-59-R (2003)

[7] Hoare, C. A. R.: An axiomatic basis for computer programming. Commun. ACM 12(10) (1969) 576-580

[8] Hoare, C. A. R.: Proof of Correctness of Data Representations. Acta Informatica 1(4) (1972) 271-281

[9] ISO/TC97/SC5/WG3: Concepts and Terminology for the Conceptual Schema and the Information Base. In: J. J. van Griethuysen, (ed.) (1982)

[10] Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. 2nd edn. Prentice Hall PTR, Upper Saddle River, NJ (2002)

[11] Martin, J., Odell, J. J.: Object-Oriented Methods. A Foundation. P T R Prentice Hall, Englewood Cliffs, New Jersey (1999)

[12] Meyer, B.: Applying 'Design by Contract'. Computer 25(10) (1992) 40-51

[13] Meyer, B.: Object-Oriented Software Construction. 2nd edn. Prentice Hall, New York (1997)

[14] Mylopoulos, J., Bernstein, P. A., Wong, H. K. T.: A Language Facility for Designing Database-Intensive Applications. ACM Transactions on Database Systems 5(2) (1980) 185-207

[15] Olivé, A.: Definition of Events and their Effects in Object-Oriented Conceptual Modeling Languages. In: Conceptual Modeling - ER 2004. LNCS 3288 (2004)

[16] OMG: UML 2.0 OCL Specification. (2003)

[17] Pressman, R. S.: Software Engineering: A Practitioner's Approach. 5th edn. McGraw Hill (2001)

[18] Queralt, A., Teniente, E.: A Platform Independent Model for the Electronic Marketplace Domain. Departament de LSI, UPC, Technical Report LSI-05-9-R (2005)

[19] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, New Jersey (1991)

[20] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison Wesley Longman, Reading, Massachusetts (1999)

[21] Wieringa, R.: A Survey of Structured and Object-Oriented Software Specification Methods and Techniques. ACM Comput. Surv. 30(4) (1998) 459-527