On the Semantics of Operation Contracts in Conceptual Modeling
Queralt, A. and Teniente, E.
Research Report LSI-04-21-R
On the Semantics of Operation Contracts in Conceptual Modeling

Anna Queralt and Ernest Teniente

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

Abstract. This paper describes two different ways of understanding operation contracts in the context of conceptual modeling. These two points of view are based in the total and partial correctness approaches from axiomatic programming and adapted to conceptual modeling. The main difference between them lies in the way postconditions and integrity constraints are guaranteed, which impacts on the desirable properties of operation contracts. Both approaches are characterized and then compared in a number of issues.

1. Introduction

The Conceptual Schema (CS) of an information system includes two kinds of knowledge: about its domain and about the functions it must perform. The former is called the Domain Conceptual Schema (DCS) and the latter the Functionality Specification (FS) [12].

A DCS 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. Integrity constraints define conditions that each state of the information base must satisfy and they may have a graphical representation or they can be defined by means of a particular language like, for instance, the OCL [13].

A FS contains a set of operations (also called domain events) and the definition of their effect on the information base. This knowledge is usually defined by pre 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 information base must satisfy after the application of the operation effect.

Operations represent the only acceptable changes of the information base, i.e. of the domain, and may be defined precisely in terms of structural events that define elementary changes in the population of an entity or a relationship type.

Several books on conceptual modeling deal both with the static and the dynamic part of a specification and give precise definitions for integrity constraints and also for pre- and postconditions [3, 6, 7, 10, 14-17]. However, they tend to pay little attention to the precise semantics of operation contracts, in the sense they that do not establish
in general an explicit relation between postconditions in the operations of the FS and the integrity constraints of the DCS.

For instance, some of these proposals do not clarify whether the satisfaction of an operation precondition necessarily requires the satisfaction of the postcondition. In addition, most of them do not state precisely which is the effect of the integrity constraints with regards to the execution of the operations and the requirements stated by the postconditions of these operations. Therefore, we believe that further research is needed to define precise semantics of operation contracts in conceptual modeling.

In this paper we describe two different approaches to understand operation contracts in the context of conceptual modeling. The first one is based on the total correctness approach from axiomatic computer programming and it has already been followed by [10]. The second approach is proposed in this paper and it is inspired in the partial correctness approach instead of the total correctness one. The main difference between them lies in the way postconditions and integrity constraints are guaranteed, which impacts on the desirable properties of requirements specifications. Both approaches are characterized and then compared in a number of issues from the point of view of the characteristics of a good software specification.

This paper is structured as follows. Next section reviews related work dealing with semantics of operation contracts in conceptual modeling. Section 3 presents an example, based on the internet auctions domain, which will be used throughout the paper. Section 4 defines total and partial correctness semantics of operation contracts. Section 5 provides a comparison of these two approaches from the point of view of the characteristics of a good software requirements specification. Finally, we present our conclusions in section 6.

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 use to pay little attention to the precise semantics of operation contracts, in the sense of the close relation between pre and postconditions and the one they both have with integrity constraints.

On the one hand, in most cases it is not clear whether or not the satisfaction of the precondition necessarily implies the satisfaction of the postcondition. For instance, in [1] we find that preconditions are relevant suppositions about the state of the objects of the information base before the execution of an operation. Postconditions describe the state of the objects of the information base after the operation completes. However, it is not clear from this definition whether operations always complete or not and, in this case, which are the reasons for not completing. Similar definitions, which share the same inconveniences than the previous one, are given in [7, 14, 16].

There are, however, other authors that define the relationship between pre- and postconditions more strongly, since they state that the postcondition is guaranteed provided the precondition holds [7, 10, 15, 17]. This means that every time an operation starts its execution, it finishes in a state satisfying the postcondition.
On the other hand, integrity constraints are usually kept separated from the discussion about operation contracts. In [7, 14-17] 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 [6] it is said explicitly that integrity constraints are not included in the discussion about pre- and postconditions for the sake of simplicity.

However, in [5, 10], the relationship between operation contracts and integrity constraints is clearly established. The implementation of every operation must satisfy both the postcondition and the integrity constraints specified in the conceptual schema every time it is executed.

3. Internet Auctions Domain

We present a conceptual schema that models the domain of internet auctions, which will serve as an example throughout the paper. More concretely, we are interested in the placing of bids for an auctioned item.

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.

Figure 1 shows a UML class diagram specifying this information. We need some additional integrity constraints in order to make the model consistent. Some of them are creation time constraints [11], which are checked only when an instance is created. Constraints are expressed in OCL, following the proposal of [11].

![Class diagram for internet auctions](image)

**Fig. 1.** Class diagram for internet auctions
Integrity constraints

- Users are identified by e-mail
  context User::emailIsUnique(): Boolean
  post: result = User.allInstances()->isUnique(email)

- Items are identified by code
  context Item::codeIsUnique(): Boolean
  post: result = Item.allInstances()->isUnique(code)

- An item may not have two open auctions simultaneously
  context Item::oneOpenAuction(): Boolean
  post: result = self.auction->one(oclIsTypeOf(OpenAuction))

- A bid can only be placed by a registered user (creation time constraint)
  context Bid::registeredUser(): Boolean
  post: result = not self.user.oclIsTypeOf(UnregisteredUser)

- A bid can only be placed for an open auction (creation time constraint)
  context Bid::auctionIsOpen(): Boolean
  post: result = self.auction.oclIsTypeOf(OpenAuction)

- The owner of a product can not place bids for it
  context Bid::bidderNotOwner(): Boolean
  post: result = self.user <> self.auction.item.user

- The amount of a bid must be greater than the initial price of the item demanded
  context Bid::amountAboveInitialPrice(): Boolean
  post: result = self.amount > self.auction.initialPrice

- The amount of a bid must be greater than the amount of all the previous bids of the same auction
  context Bid::amountAbovePreviousBids(): Boolean
  post: result = self.auction.bid->select(time.instant < self.time.instant)->forall(amount < self.amount)

We assume that the system provides an operation bid that, given a user, an amount and an auction, registers a bid for the specified auction, placed by the indicated user and with the specified amount. As usual, this operation must be specified by means of a contract, consisting of a precondition and a postcondition.

Intuitively, a precondition is an assertion involving the state previous to the execution of the operation and expresses conditions that must be satisfied when the call is done. On the other hand, a postcondition is a description of the state after the execution of the operation.

We must postpone the specification of bid to next section, since its formalization depends on the particular semantics we assume for operation contracts.

4. Semantics of Operation Contracts

As usual, we assume that an operation contract consists of a precondition and a postcondition. The precondition expresses requirements that any call must satisfy in order to be correct, while the postcondition expresses properties that are ensured in return by the execution of the call [9].

From the point of view of axiomatic computer programming, in which a precondition and a postcondition are associated to a program in order to prove its correctness,
if the precondition is true before its initiation, then it is guaranteed that the postcondition will be true on its completion [4]. Classically, there exist two approaches to state the correctness of a program with regards to its pre- and postcondition, named total correctness and partial correctness. We will give the formal definitions of each approach and then interpret their meaning from the point of view of conceptual modeling.

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 [5]. From the point of view of conceptual modeling, invariants capture semantic properties of the domain and are represented by means of integrity constraints that characterize a state of the conceptual schema. In this sense, those constraints express global properties of the state and, thus, they must be preserved by all the operations. Then, since any operation must satisfy integrity constraints on exit, integrity constraints can be considered as assertions implicitly added to the preconditions and postconditions of operation contracts [10]. The semantics of operation contracts in conceptual modeling we identify in this paper differ mainly on the way integrity constraints are considered.

4.1. Total Correctness

From the point of view of axiomatic programming, the classical definition of total correctness is as follows:

Definition 1: A program fragment is totally correct if, and only if, whenever it is executed in any state in which its precondition is satisfied, then the execution terminates and the resulting state satisfies the postcondition [8].

Since, as we have just seen, integrity constraints (i.e. invariants) must be preserved by the implementation of each operation, we have that a program is totally correct if, assuming the precondition and integrity constraints hold before its execution, it terminates and the resulting state satisfies both the postcondition and the integrity constraints. We can give an alternative definition for total correctness, this time adapted to conceptual modeling [9, 10] follow this approach to define the semantics of operation contracts.

Definition 2: An operation contract follows a total correctness semantics if its implementation guarantees that the resulting state of the information base always satisfies both the postcondition and the integrity constraints, assuming that the precondition and integrity constraints hold before its execution.

From the point of view of conceptual modeling, assuming a total correctness semantics for operation contracts does not necessarily prevent contracts from being unambiguous, which, as stated in the IEEE Recommended Practice for Software Requirements Specifications (SRS) [11], is one of the characteristics of a good SRS.

An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation [11]. This excludes the possibility of having the specification of an operation contract admitting more than one implementation. By implementation we
mean the set of structural events that satisfy the operation postcondition whenever the precondition is satisfied.

In general, the problem arises when an implementation must be chosen between several alternatives satisfying the postcondition and the integrity constraints. A contract of this kind is not acceptable, since having several implementation alternatives is a particular case of ambiguity which must be avoided.

We illustrate this problem using the contract of the operation \textit{bid} as an example. In this contract we assume that the operation has no precondition, and the postcondition asserts that a new bid is created for the given user and open auction:

\textbf{Operation:} \hspace{1em} \textit{bid}(u:User, a:OpenAuction, amount:Money)

\textbf{Precondition:}

\textbf{Postcondition:} \hspace{1em} b.oclIsNew() and b.oclIsTypeOf(Bid) and b.user = u and b.auction = a and b.amount = amount

\textbf{Table 1. Possible integrity constraint violations and their corresponding solutions}

<table>
<thead>
<tr>
<th>Violated integrity constraint</th>
<th>Possible ways of avoiding violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>registeredUser</td>
<td>• User ( u ) must become a registered user again</td>
</tr>
</tbody>
</table>
| amountAboveInitialPrice       | • \textit{Amount} should be increased  
|                               | • The initial price of the auction should be decreased |
| amountAbovePreviousBids       | • \textit{Amount} should be increased  
|                               | • The amounts of previous bids should be decreased  
|                               | • The previous bids should be eliminated |
| bidderNotOwner                | • The owner should be substituted for another user that has not participated in the auction |

As can be drawn from the table, the previous operation contract admits six different implementations with regards to total correctness semantics (one for each combination of possible ways of avoiding violation). For instance, one such implementation would consist of classifying \( u \) as \textit{User}, decreasing the initial price of the auction to a value lower than \textit{amount}, removing previous bids with a value greater than \textit{amount}, re-
placing the owner of the item by a user different to $u$ and creating a new bid identified by $u$, $a$ and the current time.

Hence, the previous operation contract does not satisfy the IEEE Recommended Practice for Software Requirements Specifications (SRS) [1] and, thus, it is not acceptable from the point of view of conceptual modeling. In order to avoid this ambiguity, the only option we have to get an acceptable contract as far as SRS is concerned is to strengthen its precondition by disambiguating the treatment of integrity constraints $\textit{amountAboveInitialPrice}$ and $\textit{amountAbovePreviousBids}$.

**Operation:** $\text{bid}(u: \text{User}, a: \text{OpenAuction}, \text{amount: Money})$

**Precondition:**
- $\text{amount} > a.\text{initialPrice}$
- $a.\text{bid}\rightarrow\text{forall}(b \mid b.\text{amount} < \text{amount})$

**Postcondition:** $b.\text{oclIsNew()}$ and $b.\text{oclIsTypeOf(Bid)}$ and $b.\text{user} = u$ and $b.\text{auction} = a$ and $b.\text{amount} = \text{amount}$

**Fig. 3.** Contract requiring unexpected actions regarding total correctness semantics

Now, as we see from this contract and the information provided in Table 1, there is a unique implementation option that satisfies both the postcondition and the integrity constraints, assuming that the precondition is also satisfied: classifying $u$ as $\text{User}$, replacing the owner of the item by a user different to $u$ and creating a new bid identified by $u$, $a$ and the current time.

One drawback of this solution is that specifying integrity constraints in the preconditions and postconditions of operations makes them unnecessary in the class diagram, which is not desirable [10]. In the previous example, this is true for the integrity constraints $\textit{amountAboveInitialPrice}$ and $\textit{AmountAbovePreviousBids}$.

Moreover, when trying to guarantee the postcondition and the integrity constraints, the effects of the operation may not necessarily be the ones desired by the designer. In other words, some implementations (properly according to total correctness semantics) may require performing unexpected actions in order to satisfy both the postcondition and the integrity constraints, as happens in the previous example with the classification of $u$ and the replacement of the owner.

To avoid this additional drawback, the only option is to strengthen the precondition adding again redundant checks that are already guaranteed by the integrity constraints.

Probably, the designer intended behaviour for the operation bid would be the one specified by the following contract according to the total correctness approach. Note that, in this case, all the requirements stated by the preconditions are redundant with regards to the ones stated by the integrity constraints of the DCS.

**Operation:** $\text{bid}(u: \text{User}, a: \text{OpenAuction}, \text{amount: Money})$

**Precondition:**
- $\text{not u.oclIsTypeOf(RegisteredUser)}$
- $\text{amount} > a.\text{initialPrice}$
- $a.\text{bid}\rightarrow\text{forall}(b \mid b.\text{amount} < \text{amount})$
- $u <> a.\text{item.user}$

**Postcondition:** $b.\text{oclIsNew()}$ and $b.\text{oclIsTypeOf(Bid)}$ and $b.\text{user} = u$ and $b.\text{auction} = a$ and $b.\text{amount} = \text{amount}$

**Fig. 4.** The operation $\textit{bid}$ regarding total correctness semantics
4.2. Partial Correctness

From the point of view of axiomatic programming, the classical definition of partial correctness is as follows:

**Definition 3:** A program fragment is partially correct if, and only if, whenever it is executed in any state in which its precondition is satisfied and its execution terminates, the resulting state satisfies the postcondition [8].

In other words, a program is partially correct if, assuming the preconditions and invariants hold before its execution, the resulting state satisfies both the postcondition and the invariants in case the execution terminates.

As we can see from definition 3, the partial correctness approach ensures the satisfaction of the postcondition when the program terminates, as total correctness does. The difference between them is that total correctness also guarantees termination. In axiomatic computer programming, it is usually said that failure to terminate may be due to an infinite loop or to the violation of an implementation limit [4].

Extending these ideas to conceptual modeling, we could regard also the violation of an integrity constraint as a reason for non-termination. In fact, integrity constraints are conditions that each state of the information base should satisfy. Hence, since operation contracts are the only means to make changes on this state, we could naturally assume that the violation of an integrity constraint prevents the operation execution from satisfying the postcondition.

As we have already seen in section 4.1, if we assume total correctness, the implementation of an operation must ensure that both the postcondition and the integrity constraints are satisfied at the end of every operation execution. However, if we understand contracts from the point of view of partial correctness, this is not necessarily true, since partial correctness does not require the satisfaction of the postcondition every time the preconditions hold. In this sense, partial correctness assumes that contracts specify the effects of operations in those cases when nothing goes wrong, i.e. when there is no violation of integrity constraints. A formalization of this idea is given in the following definition:

**Definition 4:** An operation contract follows a partial correctness semantics if its implementation checks that no integrity constraint is violated in the state satisfying the postcondition, assuming that the preconditions and integrity constraints hold before its execution. Otherwise, the operation must be rejected.

According to partial correctness, it is the programmer’s responsibility to ensure the correct behavior in the implementation of operations, taking into account the postconditions of the contracts and the integrity constraints specified in the conceptual schema. Since termination is not required, the implementation of the operation should check the integrity constraints within its code, and return an error (as well as reject the operation execution) if some of them are violated by the parameters.

As an example, consider the following operation contract is specified for the operation `bid;`
Operation: \( \text{bid}(u: \text{User}, a: \text{OpenAuction}, \text{amount:Money}) \)

Precondition:

Postcondition: \( b.\text{oclIsNew}() \) and \( b.\text{oclIsTypeOf(Bid)} \) and \( b.\text{user} = u \) and \( b.\text{auction} = a \) and \( b.\text{amount} = \text{amount} \)

\textbf{Fig. 5.} The operation \textit{bid} regarding partial correctness semantics

According to partial correctness, the same violations of integrity constraints as with total correctness may occur while trying to reach the postcondition: \textit{registeredUser, amountAboveInitialPrice, amountAbovePreviousBids} and \textit{bidderNotOwner}.

However, according to partial correctness, the semantics of the previous contract entails that this operation will not be executed (and the state of the information base will remain unchanged in order not to violate the integrity constraints) if the state of the information base where it is applied does not satisfy some of the previous conditions. Hence, the behaviour of this operation assuming partial correctness will be exactly the same as the one of the last operation contract (see Figure 4) we have specified when illustrating total correctness.

As an additional example, the behaviour of the second operation contract we have specified when illustrating total correctness (see Figure 3) should be specified as follows according to partial correctness:

Operation: \( \text{bid}(u: \text{User}, a: \text{OpenAuction}, \text{amount:Money}) \)

Precondition:

Postcondition: \( b.\text{oclIsNew}() \) and \( b.\text{oclIsTypeOf(Bid)} \) and \( b.\text{user} = u \) and \( b.\text{auction} = a \) and \( b.\text{amount} = \text{amount} \) and \\
\( \text{not } u.\text{oclIsTypeOf(UnregisteredUser)} \) and \\
\( a.\text{item.user} \neq u \)

\textbf{Fig. 6.} A different behaviour of \textit{bid} regarding partial correctness semantics

Roughly, to summarize, we could say that total correctness implicitly entails a reactive behaviour of the operations, since it assumes its implementation must take care of maintaining integrity constraints whenever they are violated so that the operation will always be executed if its precondition is satisfied. Instead, partial correctness assumes a passive behaviour of the operations, since it assumes its implementation must check integrity constraints and reject the operation execution if some constraint is violated.

5. Discussion

We will compare the two approaches from the point of view of the relevant characteristics of a good software requirements specification [1, 2]. As recommended also in [1, 2], we assume an unambiguous operation contract specification.
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. Total correctness avoids erroneous execution of an operation by means of its precondition while partial correctness 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: the operation has no effect on the information base.

For instance, understanding the operation bid from the point of view of total correctness (figure 4), we always obtain a defined result when the precondition is satisfied, which is exactly the one specified in the precondition (a new bid is created and associated to the user and auction specified as parameters) plus the additional changes of the information base required to satisfy all violated integrity constraints.

Understanding bid from partial correctness (figure 5), we have two kinds of results when the precondition holds. In these cases where no integrity constraint is violated, the resulting state of the information base is the one specified in the precondition, as occurs in total correctness. 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 inconsistency, partial correctness facilitates having a consistent specification, while total correctness is more prone to inconsistencies. The reason is that, since integrity constraints are sometimes specified both in the DCS and as preconditions of operations in the FS, 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 4 than with the one in figure 5, 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 total correctness. Although both approaches allow the verification of the software product, this process can become more complicated assuming partial correctness due to the dispersion of the requirements affecting an operation.

For example, to know whether the contract in figure 5 specifies correctly the meaning of a bid in an Internet auction, we must also take into account the integrity constraints registeredUser, amountAboveInitialPrice, amountAbovePreviousBids and bidderNotOwner. Taking, however, the contract in figure 4, no additional information is needed in order to check whether it meets the requirements.

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 total correctness 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 total correctness we will also have to modify the precondition of the operation bid (see the contract in figure 4), 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 partial correctness, we need not need to make any additional changes, since requirements stated by integrity constraints are stated only in the DCS.

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, partial correctness needs to specify this reactive behaviour always in the postcondition. On the other, total correctness requires to do the same to prevent ambiguous reactions to the violation of integrity constraints.

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 partial correctness 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 4 and 5. It is clear than the one in figure 5, corresponding to partial correctness, is much shorter and, however, both of them have the same meaning.
5.6. Transition to design and implementation

Finally, we analyse whether total and partial correctness facilitate the transition from the operations' specification to the following stages of software development, i.e. to design and implementation.

From the point of view of total correctness, since integrity constraints are specified again in the preconditions of the contracts, a certain aspect could be designed and implemented twice, in two different ways, if the designer or programmer does not realise that it is redundantly specified.

For instance, if we directly design the transaction corresponding to the contract of \( bid \) shown in Figure 4, it will need to verify each of the preconditions of this contract. However, if we are implementing this system in a relational database, it may happen they are already guaranteed by the DBMS due to primary keys (as with constraint \( \text{register\_dUser} \) or by means of triggers (as we could do with the other constraints).

This is less probable with partial correctness, since each requirement is only specified in a single place. Nevertheless, with this approach we require an additional design step to include checks in the design of the operations for those integrity constraints of the DCS that are not directly guaranteed by the DBMS itself.

There is an additional advantage if total correctness is assumed. Since all conditions in which an operation is not valid are specified in the precondition of its contract, the designer has all the information at the same place and does not need to search for additional conditions in the DCS. Clearly, this is not the case when assuming partial correctness.

The following table summarises the discussion in this section.

<table>
<thead>
<tr>
<th></th>
<th>Total Correctness</th>
<th>Partial Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Consistency</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Verifiability</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Modifiability</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Conciseness</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Transition</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

6. Conclusions

The main goal of this paper has been to clarify the semantics of operation contracts in conceptual modeling. In this sense, we have identified and characterized two different approaches to understand operation contracts of the FS in relation to the integrity constraints defined in the DCS. These two approaches are inspired in the total and partial correctness approaches from axiomatic computer programming and have been adapted to conceptual modeling.

The total correctness approach \( \beta, 10 \) states that an operation is correct if, assuming the precondition and integrity constraints hold before its execution, it is guaran-
need that it terminates and the resulting state satisfies both the postcondition and the integrity constraints.

The partial correctness approach, as proposed in this paper, states that an operation is correct if, whenever it is executed in any state in which its precondition and the integrity constraints are satisfied, it terminates only if no integrity constraint is violated in the state that satisfies the postcondition. Otherwise, the operation is rejected.

We have also compared both approaches in a number of issues from the point of view of the characteristics of a good software specification and of their contribution to software development: completeness, consistency, verifiability, modifiability, conciseness and transition to software design and implementation. The results of this comparison have been summarized in Table 2.

From our discussion, we may conclude that partial correctness does better than total correctness as far as consistency, modifiability and conciseness of the conceptual schema are concerned; total is better than partial regarding verifiability, and they both present similar contributions with respect to completeness and facilitation of the transition to design and implementation.

Acknowledgements

We would like to thank Jordi Cabot, Jordi Conesa, Óscar Hernández, Antoni Olivé and Xavier de Pau for helpful discussions and comments on previous drafts of this paper, and Fernando Orjas and Robert Nieuwenhuis for their clarifications.

This work has been partially supported by the Ministerio de Ciencia y Tecnología and the FEDER funds, under the project TIC2002-00744.

References

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Research Reports - 2004

- LSI-04-2-R: *Comparison of Methods to Predict Ozone Concentration*, Orozco, J.
- LSI-04-3-R: *Towards the definition of a taxonomy for the cots product’s market*, Ayala, Claudia P.
- LSI-04-4-R: *Modelling Coalition Formation over Time for Iterative Coalition Games*, Mérida-Campos, C. and Willmott, S.
- LSI-04-6-R: *An Analysis Pattern for Electronic Marketplaces*, Queralt, A. and Teniente, E.
- LSI-04-7-R: *Exploring Dopamine-Mediated Reward Processing through the Analysis of EEG-Measured Gamma-Band Brain Oscillations*, Vellido, A. and El-Deredy, W.
- LSI-04-15-R: *Evaluation and symmetrisation of alignments obtained with the Giza++ software*, Lambert, P. and Castell, N.
- LSI-04-16-R: *A note on the use of topology extensions for provoking instability in communication networks*, Blesa, M.J.
Hardcopies of reports can be ordered from:

Núria Sanchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Campus Nord, Mòdul C6
Jordi Girona Salgado, 1-3
03034 Barcelona, Spain
nurias@lsi.upc.es

See also the Departament WWW pages, http://www.lsi.upc.es/