Specification of the Application SuperSport with ER-DFD

Fabrizio Riguzzi

Dipartimento di Ingegneria, Università di Ferrara
Via Saragat, 1 44100 Ferrara, Italy
friguzzi@ing.unife.it

Technical Report CS-2003-01
Dipartimento di Ingegneria, Università di Ferrara

1 Introduction

This paper presents the specification of the application SuperSport expressed as an Entiry Relationship – Data Flow Diagram.

The application was inspired by the application StraSport described in chapter 14 of [GR02].

The paper is organized as follows: section 2 describes the formalism of Entity Relationship – Data Flow Diagrams and section 3 presents the application.

2 Entity Relationship - Data Flow Diagrams

Entity Relationship - Data Flow Diagrams (ER-DFD) [LMR97] are an integration of Entity Relationship diagrams [Che78] and Data Flow Diagram [DeM78] that is similar to Formal Data Flows Diagrams [FGM88]: the data stores of DFD are replaced by entities and relationships of the ER diagrams, therefore we have data flows entering directly into entities and relationships and coming out from them. Moreove, we distinguish three types of data flows: proper data flows, that represent the exchange of data, error flows, that represent the exchange of error messages, and control flows, that represent the exchange of control information.

In order to distinguish between elements of the DFD and ER diagram which have a similar graphical symbol, we adopt the following conventions: external agents (the user or other applications) of DFD are represented with a dashed line box to distinguish it from entities represented as normal boxes and data flows, error flows and control flows are represented by arrows to distinguish them from the connections between entities and relationship represented as simple lines. Data flows, error flows and control flows are distinguished on the basis of the line of the arrow: continuous for data flows, dashed for error flows and dotted for control flows. Figure 1 shows a sample diagram. In this diagram, entity1 is a weak entity: its key is formed by the attribute a1 plus the key of entity2, b1.

A number of fields are associated with each data flow: when a field has the same name of the attribute of an entity, they refer to the same data. When the field does not correspond to any attribute, it represents data derived from attributes by computation. Error flows and control flows do not have any fields associated with them.

We suppose that the diagram contains also the indication of the boundaries of the different applications in the form of dashed lines.

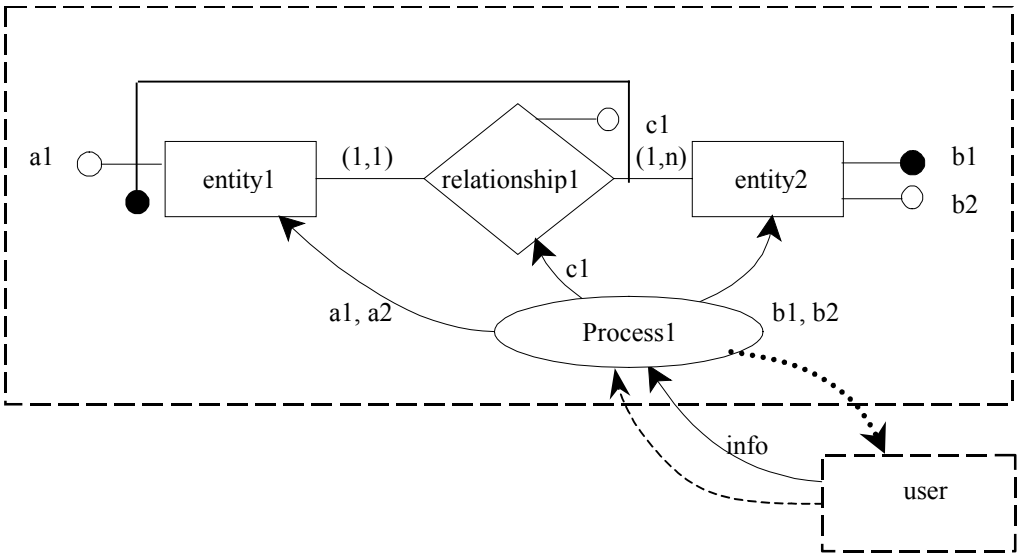


Figure 1: Example of an ER-DFD diagram.

3 Application SuperSport

SuperSport is a company that wholesales sport apparel through a network of branches. It uses the application SuperSport in order to manage its sales and the application BranchManagement in order to manage its branches.

SuperSport's clients are usually big chains of department stores that have multiple shop on the territory. Each client is served by one or more branches and by one or more agents.

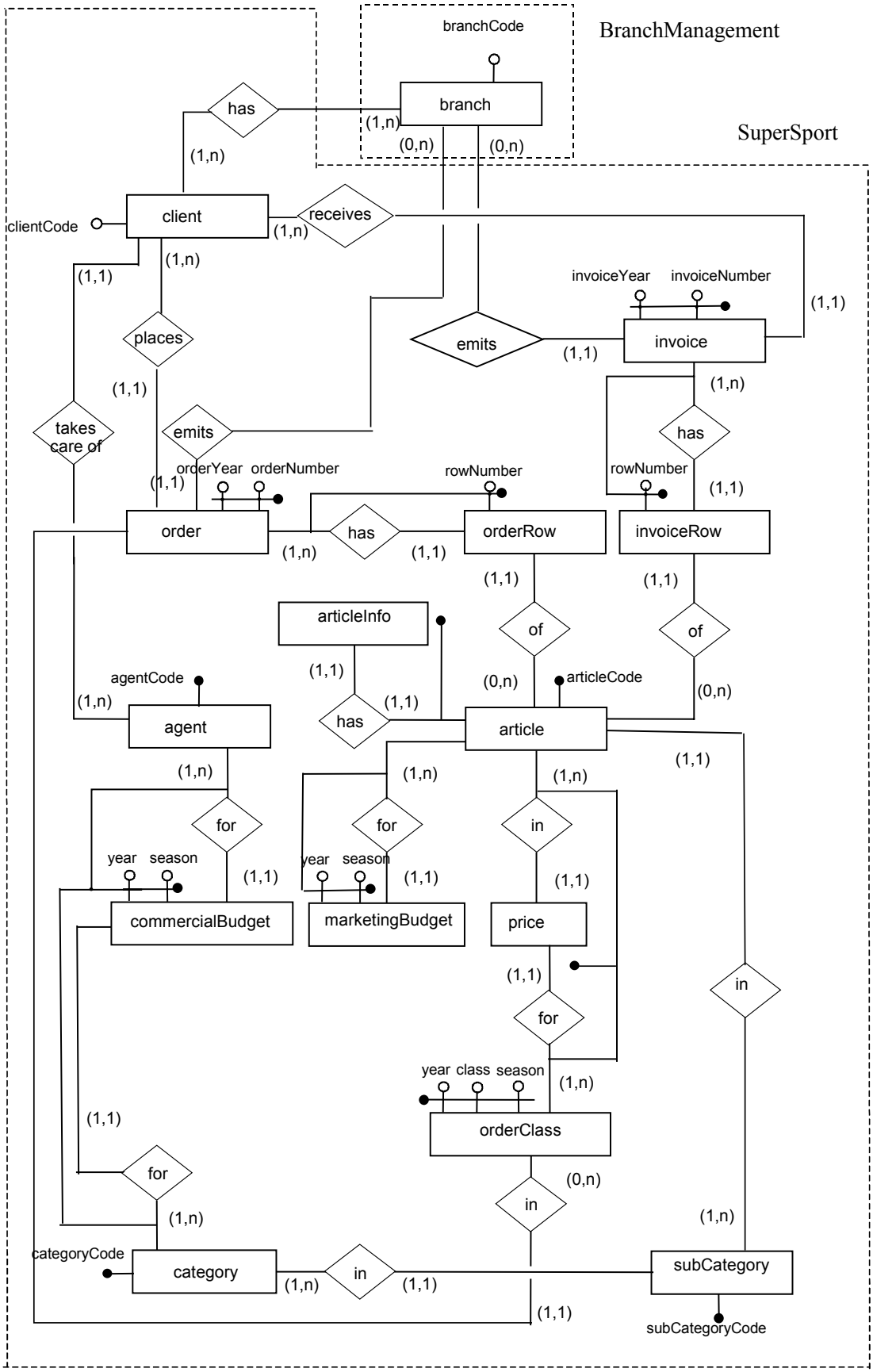
Client's orders are collected by a branch. Each order contains multiple rows, each referring to a specific article. Once the order has been fulfilled, an invoice is emitted by the competent branch. An invoice is emitted starting from a corresponding order and adding information about the quantity of articles effectively delivered and the actual price.

Each article is described by a series of info contained in articleInfo. Moreover, each article belongs to a sub-category (e.g. swimming costumes, beach costumes). Each sub-category belongs to a category.

The application SuperSport stores the prices given the article, the year, the season (spring/summer or autumn/winter) and class of the order (for example: "by phone", "by mail", etc.).

The application SuperSport is also used in order to store predictions regarding the future: the predictions of quantities sold, of annullments, of returns, etc. for each year, season, agent and category are stored in commercialBudget, while the same predictions for each year and season are stored in marketingBudget.

The Entity-Relationhip diagram of the application is shown below.



As redards the Function Point count of the application SuperSport, we assume to know that the entity branch forms an ILF for the application BranchManagement.

The attributes owned by the entities are listed below.

branch:

branch
branchCode
branchName
branchAddress
branchProvince
distributionChannel

client:

client
clientCode
name
address
zipCode
province
fiscalCode
vatNumber
region
macroArea
relationshipStartDate
relationshipEndDate
discount
endOfYearDiscount
telephoneNumber
faxNumber
clientClass
salseVolumeClass
solvency

agent:

agent
agentCode
description
commission
agency
type

order:

order
year
number
season
payment
discountOnPayment
unconditionalDiscount
paymentDueDate
orderDate
orderQuantity
paymentType

orderRow:

orderRow
year
number
rowNumber
quantity
deliveryDate
annullmentReason

article:

articleCode
description
color
model
vatCode
warehouseLocation

articleInfo:

articleCode
theme
productionYear
productionSeason
designer
averagePrice
producer
supplier
target
line
standardBuyPrice

invoice:

invoiceYear
invoiceNumber
date

invoiceRow:

invoiceYear
invoiceNumber
invoiceRowNumber
quantity
unconditionalDiscount
price

category:

categoryCode
description

subCategory:

subCategoryCode
description
categoryCode

price:

year
season
articleCode
class
price

orderClass:

year
season
class

marketingBudget:

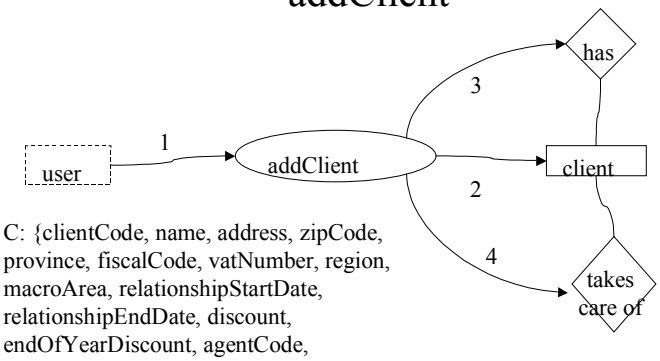
year
season
articleCode
returnsForecast
annullmentsForecast
finalYearDiscountsForecast
grossValueForecast
totalQuantityForecast
invoiceDiscountForecast

commercialBudget:

year
season
agentCode
categoryCode
returnsForecast
annullmentsForecast
discountsForecast
grossValueForecast
totalQuantityForecast

The application SuperSport contains the processes described below.

addClient

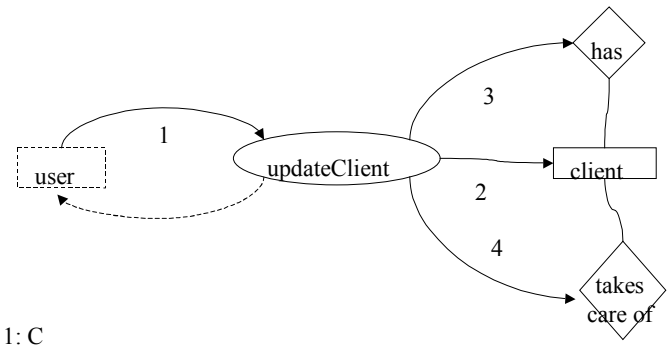


C: {clientCode, name, address, zipCode, province, fiscalCode, vatNumber, region, macroArea, relationshipStartDate, relationshipEndDate, discount, endOfYearDiscount, agentCode, telephoneNumber, faxNumber, clientClass, salseVolumeClass, solvency, branchCode}.

1: C

2: C\{agentCode,branchCode} 3: branchCode 4: agentCode

updateClient

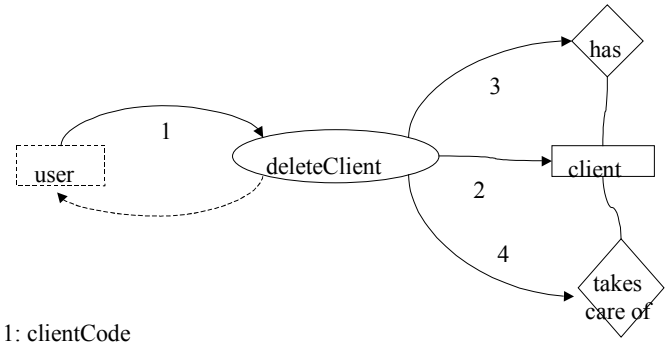


1: C

2: C\{agentCode,branchCode} 3: branchCode 4: agentCode

An error is returned if the clientCode given as input is not already present in client.

deleteClient

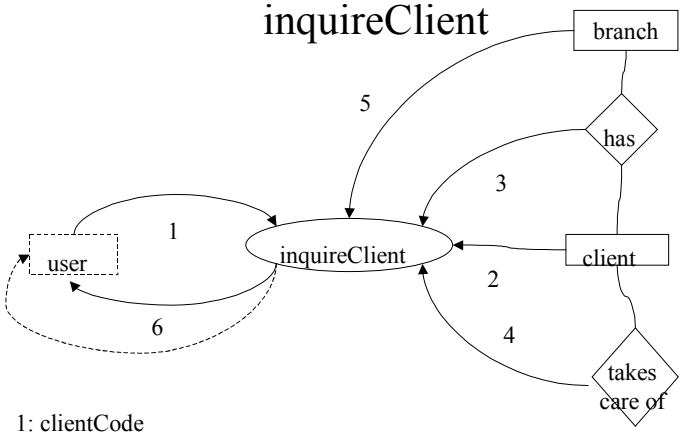


1: clientCode

2: C\{agentCode,branchCode} 3: branchCode 4: agentCode

An error is returned if the clientCode given as input is not already present in client.

inquireClient

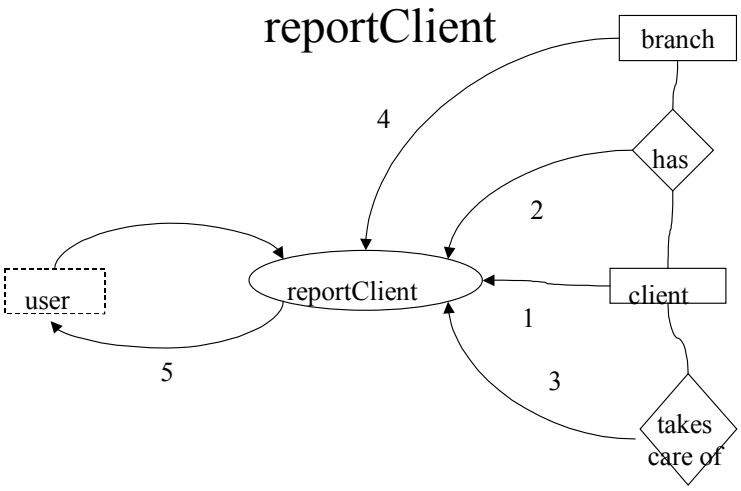


1: clientCode

2: C\{agentCode,branchCode} 3: branchCode 4: agentCode

5: branchName 6: C ∪ {branchName}

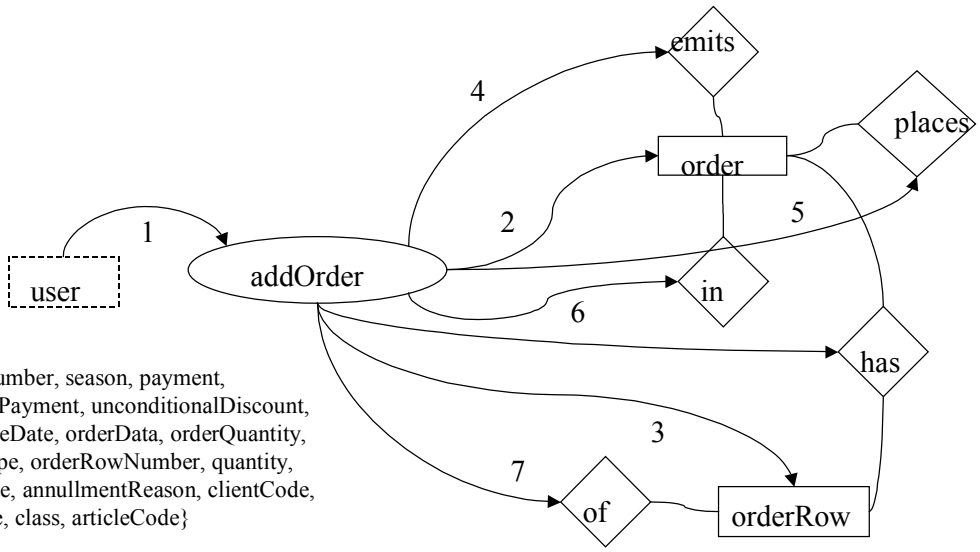
An error is returned if the clientCode given as input is not already present in client.



1: C\{agentCode,branchCode} 2: branchCode 3: agentCode

4: branchName 5: C ∪ {branchName,totalNumberOfClients}

addOrder



O: {year, number, season, payment, discountOnPayment, unconditionalDiscount, paymentDueDate, orderData, orderQuantity, paymentType, orderRowNumber, quantity, deliveryDate, annullmentReason, clientCode, branchCode, class, articleCode}

1: O

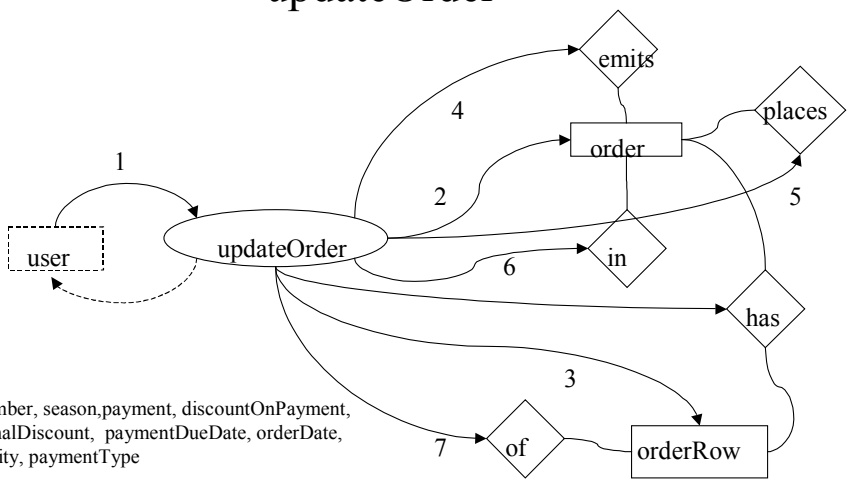
2: year,number, season, payment, discountOnPayment, unconditionalDiscount, paymentDueDate, orderDate, orderQuantity, paymentType

3: year, number, rowNumber, quantity, deliveryDate, annullmentReason

4: branchCode 5: clientCode 6: year, season, class

7: articleCode

updateOrder



1: O

2: year, number, season,payment, discountOnPayment, unconditionalDiscount, paymentDueDate, orderDate, orderQuantity, paymentType

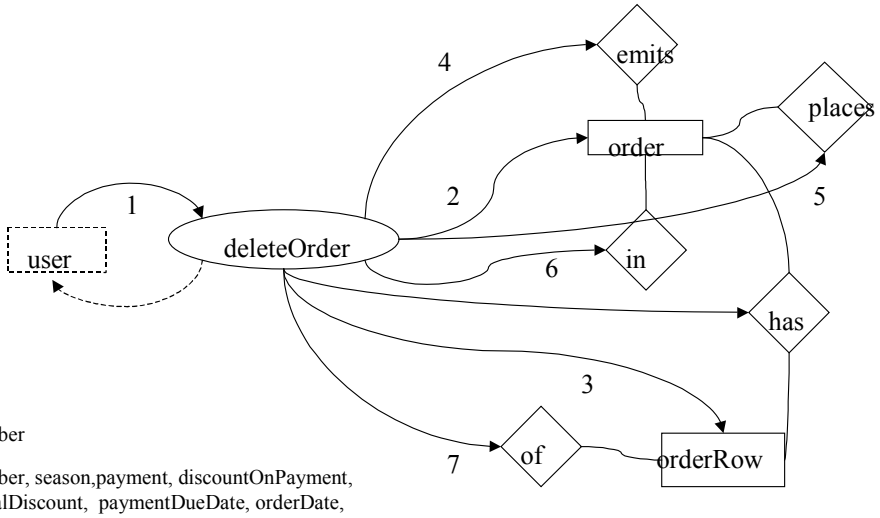
3: year, number, rowNumber, quantity, deliveryDate, annullmentReason

4: branchCode 5: clientCode 6: year, season, class

7: articleCode

An error is returned if an order with year, number given as input is not already present in order.

deleteOrder



1: year, number

2: year, number, season,payment, discountOnPayment, unconditionalDiscount, paymentDueDate, orderDate, orderQuantity, paymentType

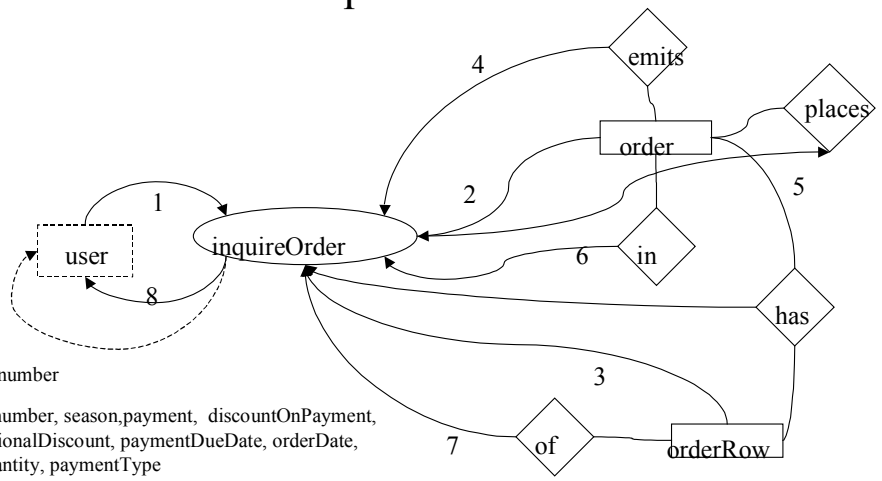
3: year, number, rowNumber, quantity, deliveryDate, annullmentReason

4: branchCode 5: clientCode 6: year, season, class

7: articleCode

An error is returned if an order with year, number given as input is not already present in order

inquireOrder



1: year, number

2: year, number, season,payment, discountOnPayment, unconditionalDiscount, paymentDueDate, orderDate, orderQuantity, paymentType

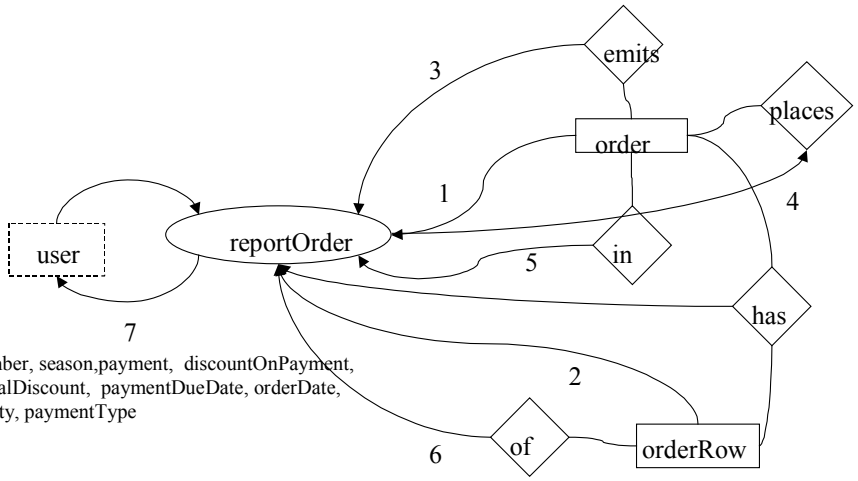
3: year, number, rowNumber, quantity, deliveryDate, annullmentReason

4: branchCode 5: clientCode 6: year, season, class

7: articleCode 8: O

An error is returned if an order with year, number given as input is not present in order

reportOrder



1: year, number, season,payment, discountOnPayment, unconditionalDiscount, paymentDueDate, orderDate, orderQuantity, paymentType

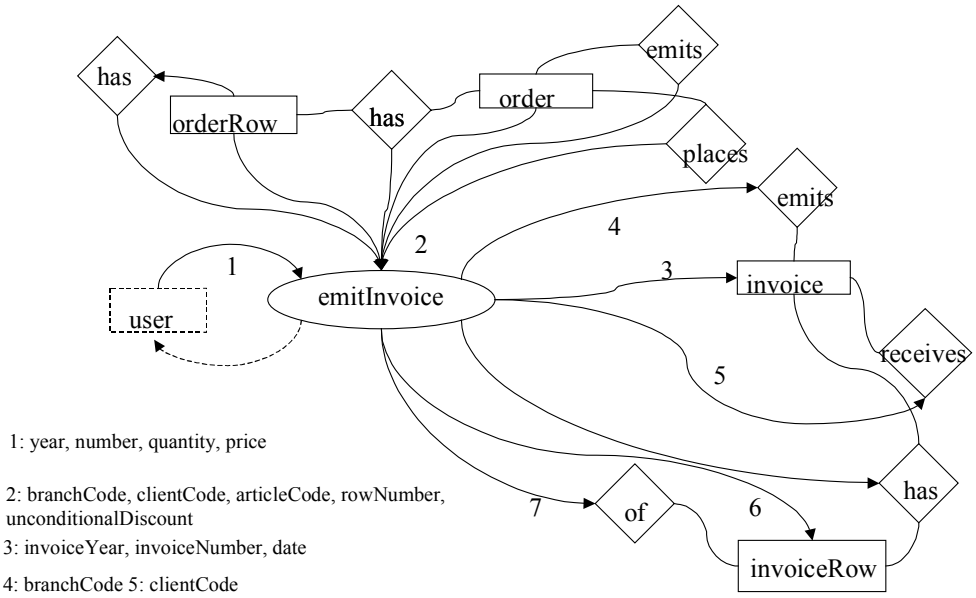
2: year, number, rowNumber, quantity, deliveryDate, annullmentReason

7: O ∪ {totalNumberOfOrders}

3: branchCode 4: clientCode 5: year, season, class

6: articleCode

emitInvoice



1: year, number, quantity, price

2: branchCode, clientCode, articleCode, rowNumber, unconditionalDiscount

3: invoiceYear, invoiceNumber, date

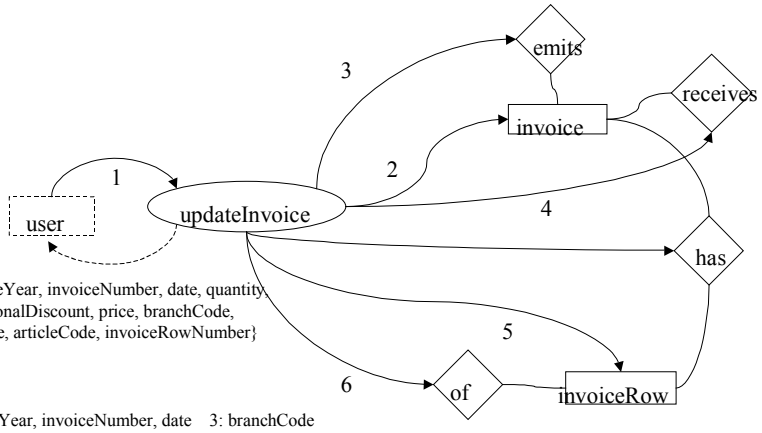
4: branchCode 5: clientCode

6: invoiceYear, invoiceNumber, invoiceRowNumber, quantity, unconditionalDiscount, price

7: articleCode

emitInvoice takes as input the order year and number and emits the corresponding invoice. Moreover, it takes as input the effective quantities and prices of articles (that may differ from those of the order). It returns an error in the case the order taken as input is not present in order.

updateInvoice



I: {invoiceYear, invoiceNumber, date, quantity, unconditionalDiscount, price, branchCode, clientCode, articleCode, invoiceRowNumber}

1: I

2: invoiceYear, invoiceNumber, date 3: branchCode

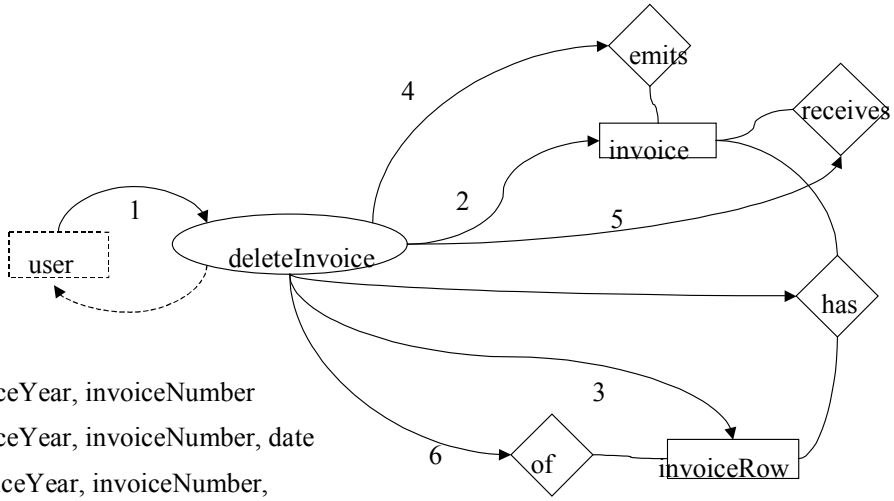
4: clientCode

5: invoiceYear, invoiceNumber, invoiceRowNumber, quantity, unconditionalDiscount, price

6: articleCode

It returns an error in the case the invoice taken as input (described by invoiceYear, invoiceNumber) does not already exist.

deleteInvoice



1: invoiceYear, invoiceNumber

2: invoiceYear, invoiceNumber, date

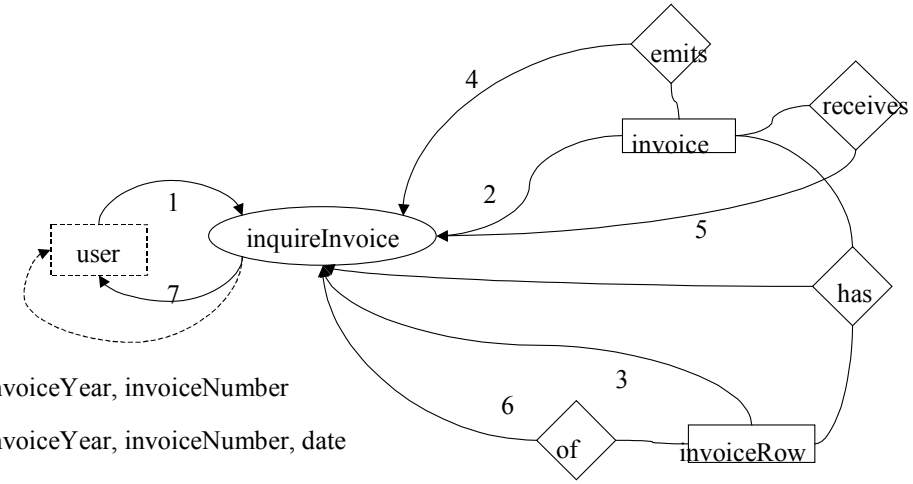
3: invoiceYear, invoiceNumber, invoiceRowNumber, quantity, unconditionalDiscount, price

4: branchCode 5: clientCode

6: articleCode

It returns an error in the case the invoice taken as input (described by invoiceYear, invoiceNumber) does not exist.

inquireInvoice



1: invoiceYear, invoiceNumber

2: invoiceYear, invoiceNumber, date

3: invoiceYear, invoiceNumber, invoiceRowNumber, quantity, unconditionalDiscount, price

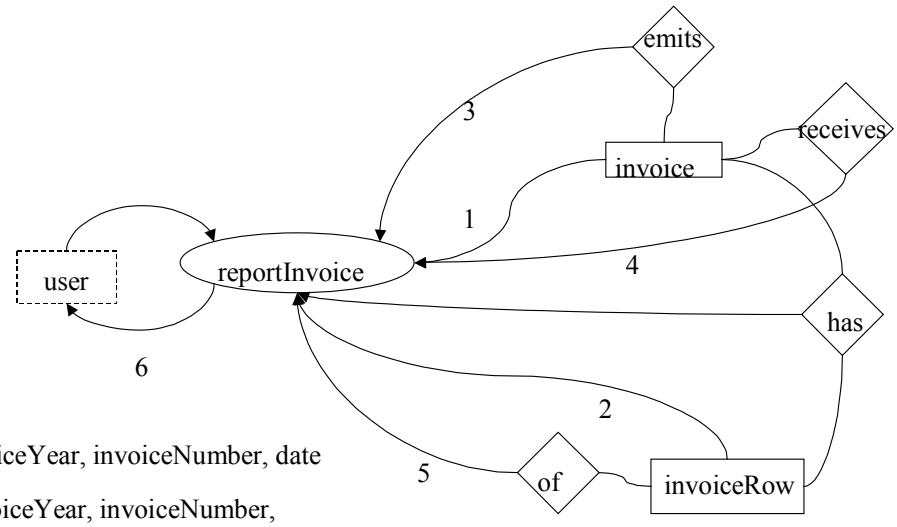
4: branchCode 5: clientCode

6: articleCode

7: I

It returns an error in the case the invoice taken as input (described by invoiceYear, invoiceNumber) does not exist.

reportInvoice



1: invoiceYear, invoiceNumber, date

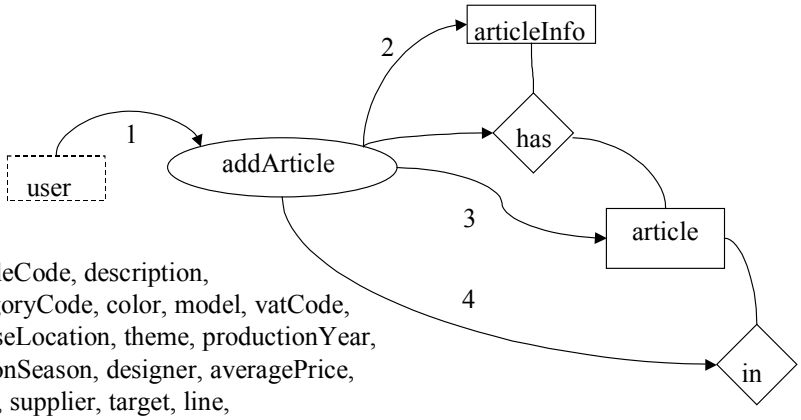
2: invoiceYear, invoiceNumber, invoiceRowNumber, quantity, unconditionalDiscount, price

3: branchCode 4: clientCode

5: articleCode

6: I ∪ {totalNumberOfInvoices}

addArticle



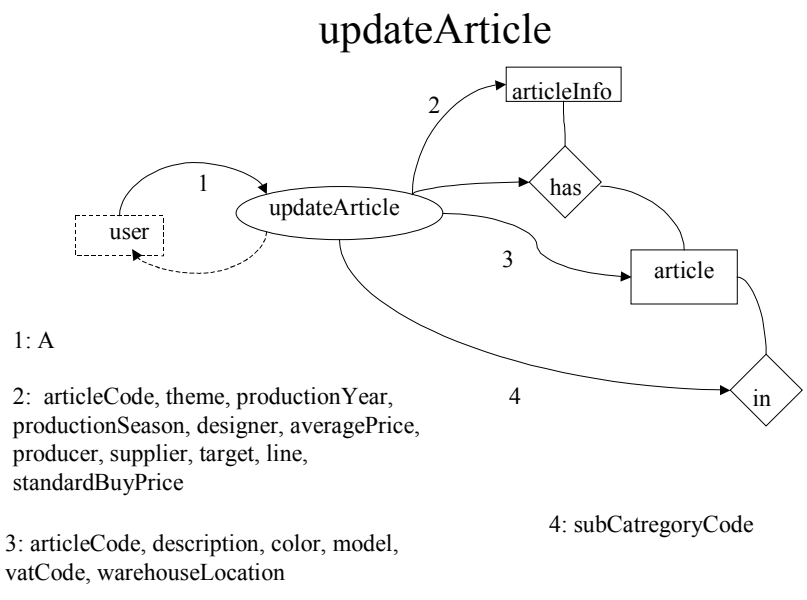
A: {articleCode, description, subCategoryCode, color, model, vatCode, warehouseLocation, theme, productionYear, productionSeason, designer, averagePrice, producer, supplier, target, line, standardBuyPrice}

1: A

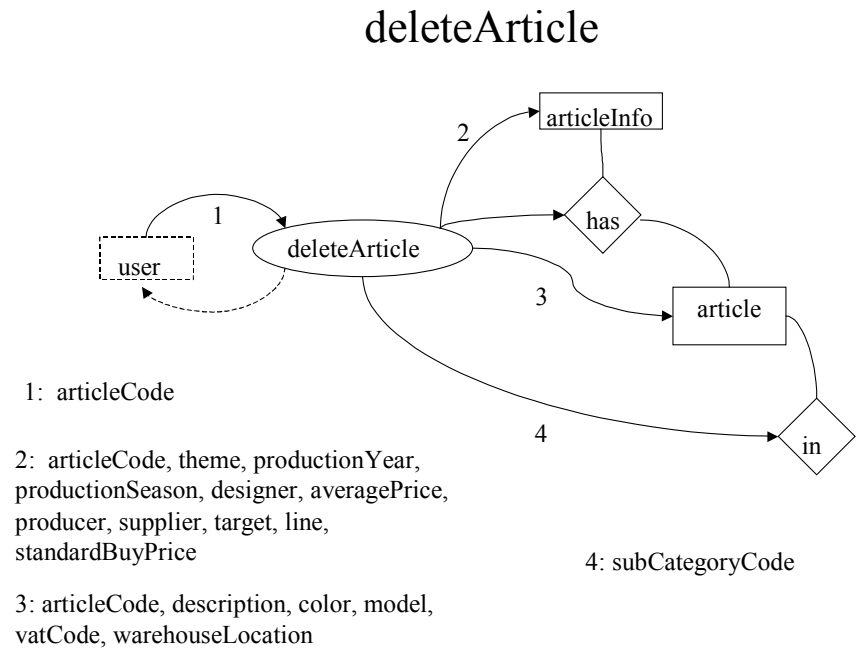
2: articleCode, theme, productionYear, productionSeason, designer, averagePrice, producer, supplier, target, line, standardBuyPrice

3: articleCode, description, color, model, vatCode, warehouseLocation

4: subCategoryCode

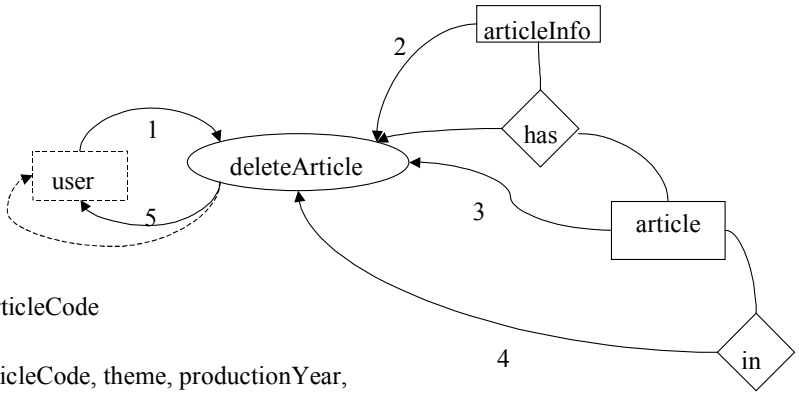


It returns an error in the case the article taken as input (described by articleCode) does not exist.



It returns an error in the case the article taken as input (described by articleCode) does not exist.

inquireArticle



1: articleCode

2: articleCode, theme, productionYear, productionSeason, designer, averagePrice, producer, supplier, target, line, standardBuyPrice

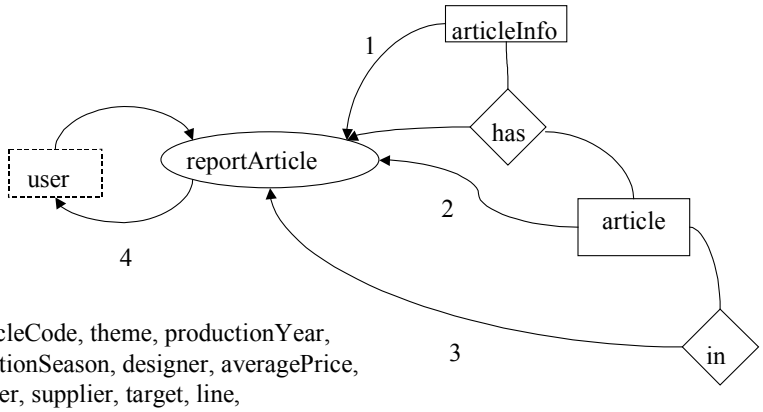
3: articleCode, description, color, model, vatCode, warehouseLocation

4: subCategoryCode

5: A

It returns an error in the case the article taken as input (described by articleCode) does not exist.

reportArticle



1: articleCode, theme, productionYear, productionSeason, designer, averagePrice, producer, supplier, target, line, standardBuyPrice

2: articleCode, description, color, model, vatCode, warehouseLocation

3: subCategoryCode

4: A∪ {totalNumberOfArticles}

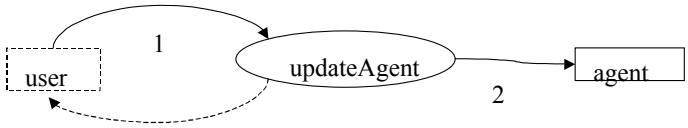
addAgent

AG: agentCode, description, commission, agency, type

1: AG

2: AG

updateAgent



1: AG

2: AG

It returns an error in the case the agent taken as input (described by agentCode) does not exist.

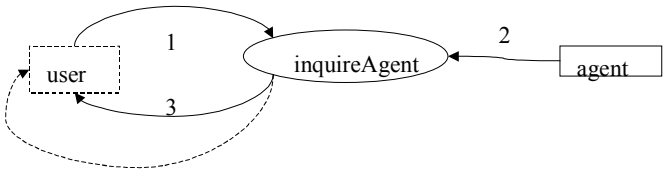
deleteAgent

1: agentCode

2: AG

It returns an error in the case the agent taken as input (described by agentCode) does not exist.

inquireAgent

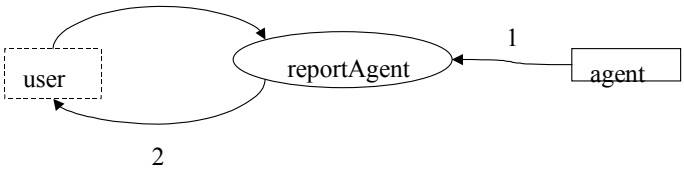


1: agentCode

2, 3: AG

It returns an error in the case the agent taken as input (described by agentCode) does not exist.

reportAgent



1: AG

2: AG ∪ {totalNumberOfAgents}

addOrderClass

OC: {year, season, class}

1: OC

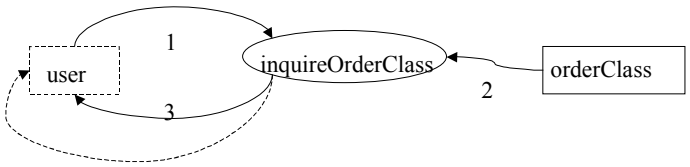
2: OC

deleteOrderClass

1: OC
2: OC

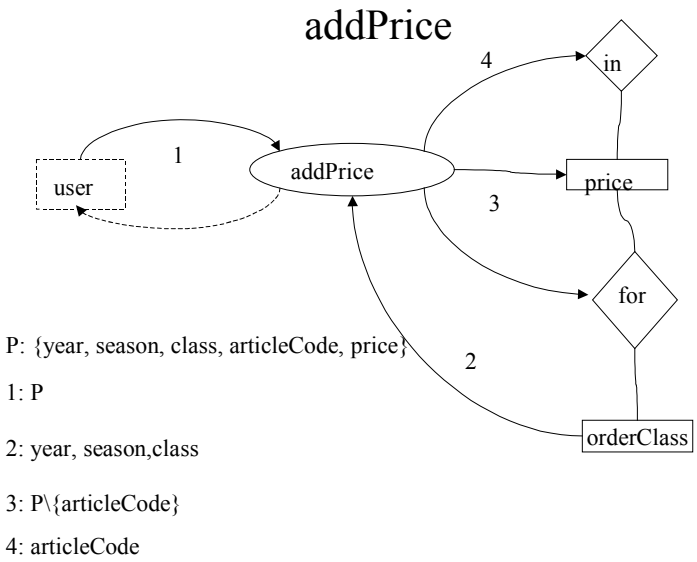
It returns an error in the case the order class taken as input (described by year, season and class) does not exist.

inquireOrderClass



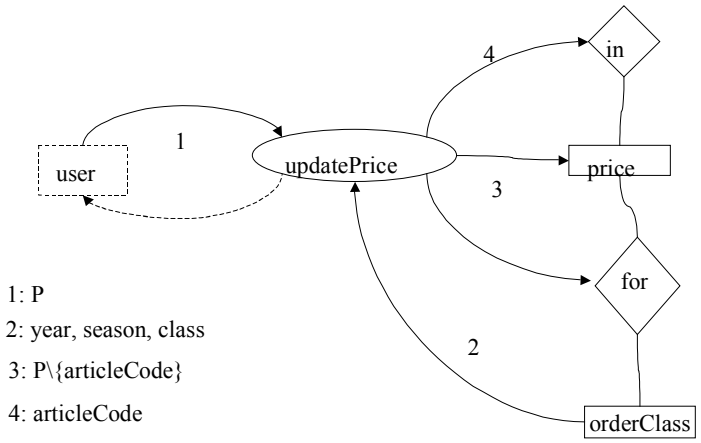
1: year, season
2, 3: OC

It returns an error in the case there is no order class for the year and season given as input.



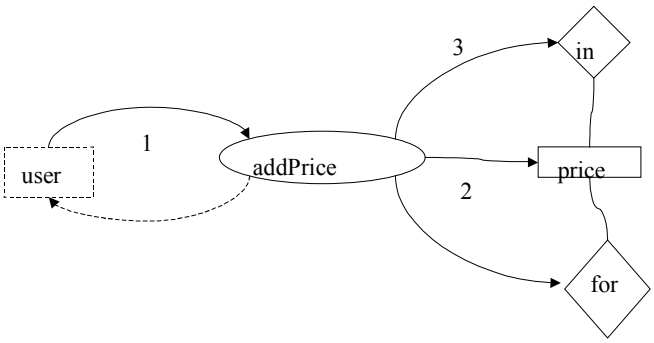
When a new price is added, the process addPrice checks that the triple year, season, class that is given as input is already present in orderClass otherwise it returns an error.

updatePrice



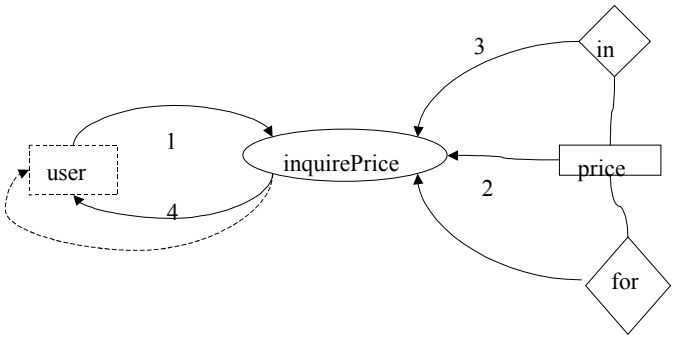
When a price is updated, the process updatePrice checks that the triple year, season, class that is given as input is already present in orderClass. In the case where year, season, class do not represent a valid class or where a record with year, season, class and articleCode does not already exist in price, an error is returned.

deletePrice



In the case where no record exist in price with the values for year, season, class and articleCode given as input, an error is returned.

inquirePrice



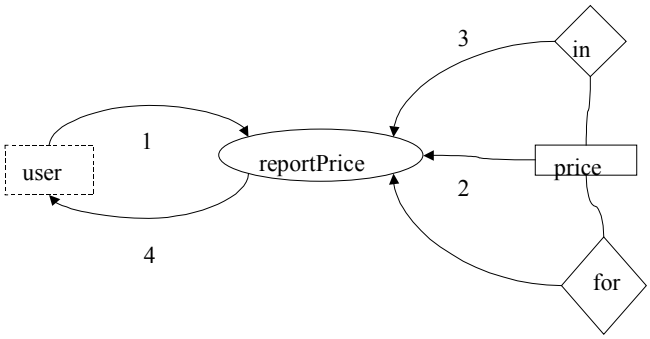
1: year, season, class, articleCode

2: P\{articleCode}

3: articleCode 4: P

In the case where no record exist in price with the values for year, season, class and articleCode given as input, an error is returned.

reportPrice

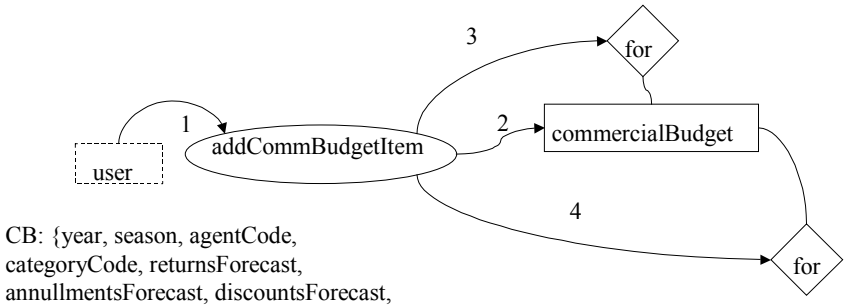


1: year, season, class

2: P\{articleCode}

3: articleCode 4: P

addCommBudgetItem



CB: {year, season, agentCode,
categoryCode, returnsForecast,
annullmentsForecast, discountsForecast,
grossValueForecast,
totalQuantityForecast}

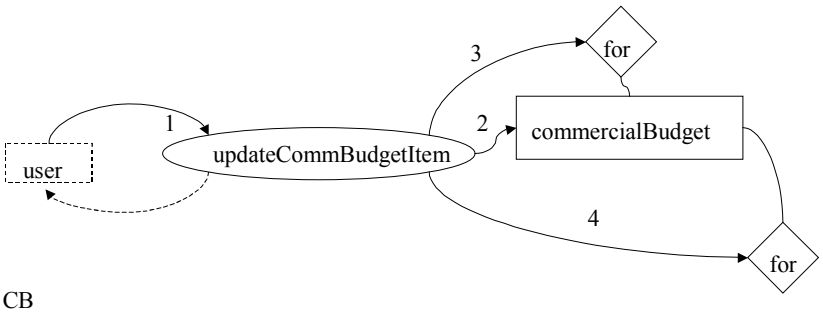
1: CB

2: CB\{agentCode,categoryCode}

3: agentCode

4: categoryCode

updateCommBudgetItem



1: CB

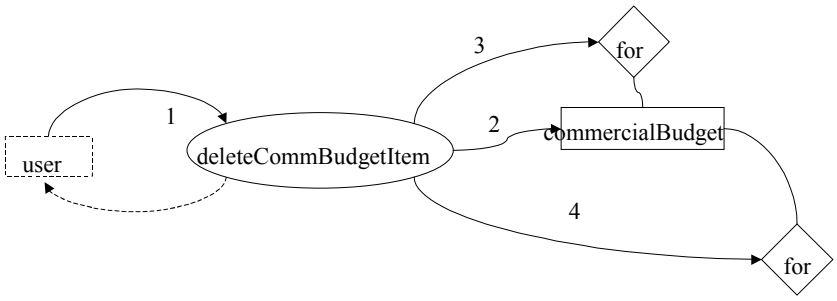
2: CB\{agentCode,categoryCode}

3: agentCode

4: categoryCode

It returns an error in the case the commercial budget item taken as input (described by year, season, categoryCode and agentCode) does not exist.

deleteCommBudgetItem



1: year, season, agentCode, categoryCode

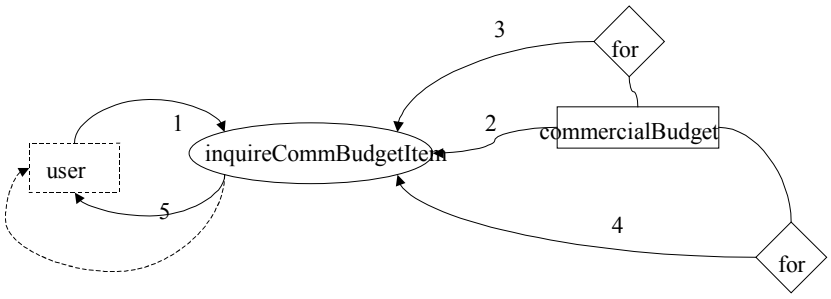
2: CB\{agentCode,categoryCode}

3: agentCode

4: categoryCode

It returns an error in the case the commercial budget item taken as input (described by year, season, categoryCode and agentCode) does not exist.

inquireCommBudgetItem



1: year, season, agentCode, categoryCode

2: CB\{agentCode,categoryCode}

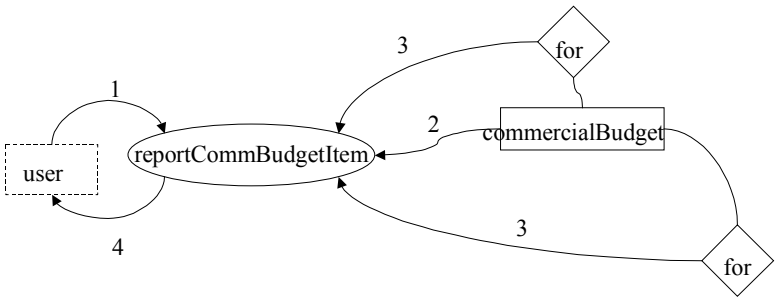
3: agentCode

4: categoryCode

5: CB

It returns an error in the case the commercial budget item taken as input (described by year, season, categoryCode and agentCode) does not exist.

reportCommBudgetItem



1: year, season

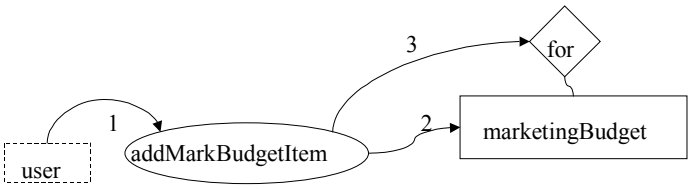
2: CB\{agentCode, categoryCode}

3: agentCode

4: categoryCode

5: CB ∪ {totalNumberOfCBItems}

addMarkBudgetItem



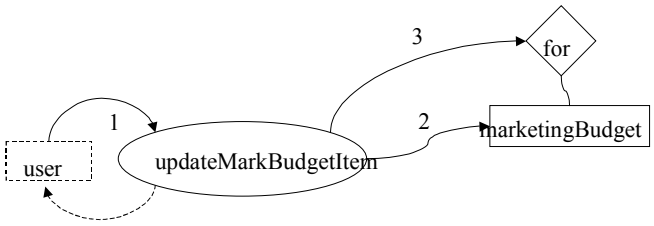
MB:{ year, season, articleCode, returnsForecast, annullmentsForecast, finalYearDiscountsForecast, grossValueForecast, totalQuantityForecast, invoiceDiscountForecast}

1: MB

2: MB\{articleCode}

3: articleCode

updateMarkBudgetItem



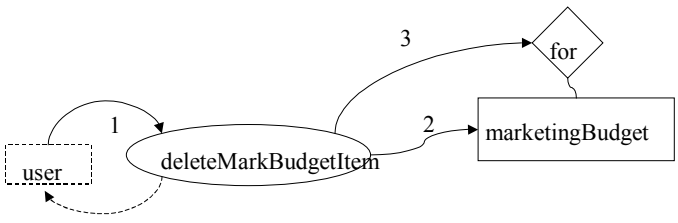
1: MB

2: MB\{articleCode}

3: articleCode

It returns an error in the case the marketing budget item taken as input (described by year, season, and articleCode) does not exist.

deleteMarkBudgetItem



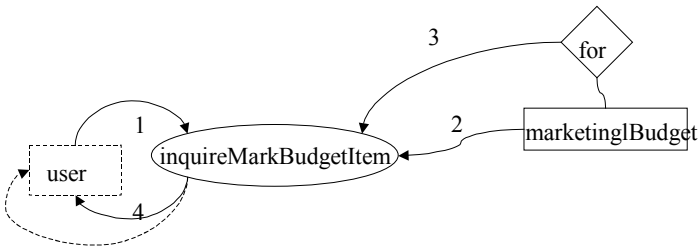
1: year, season, articleCode

2: MB\{articleCode}

3: articleCode

It returns an error in the case the marketing budget item taken as input (described by year, season, and articleCode) does not exist.

inquireMarkBudgetItem



1: year, season, articleCode

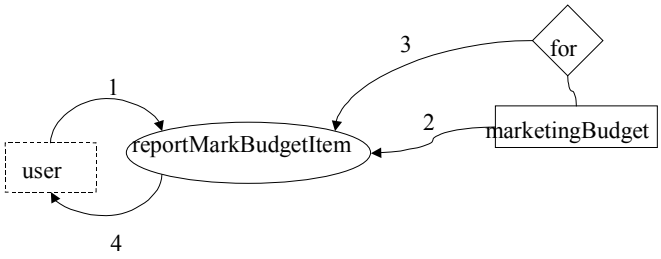
2: MB\{articleCode}

3: articleCode

4: MB

It returns an error in the case the marketing budget item taken as input (described by year, season, and articleCode) does not exist.

reportMarkBudgetItem



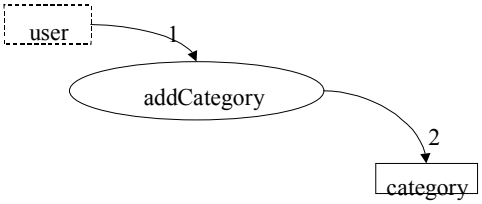
1: year, season

2: MB\{articleCode}

3: articleCode

4: MB ∪ {totalNumberOfMBItems}

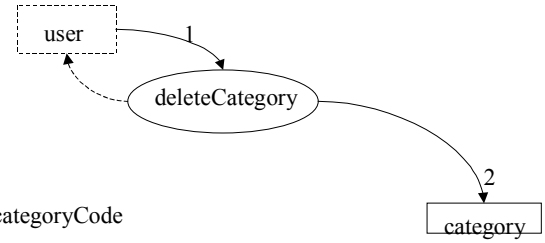
addCategory



1: categoryCode, description

2: categoryCode,
description

deleteCategory

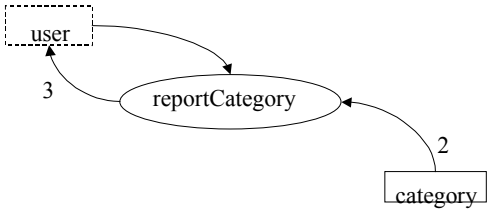


1: categoryCode

2: categoryCode,
description

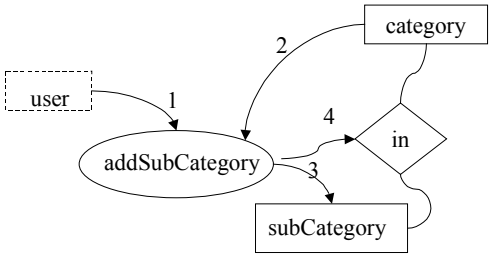
It returns an error in the case the category taken as input (described by categoryCode) does not exist.

reportCategory



2, 3: categoryCode,
description

addSubCategory



1: subCategoryCode, description,
categoryCode

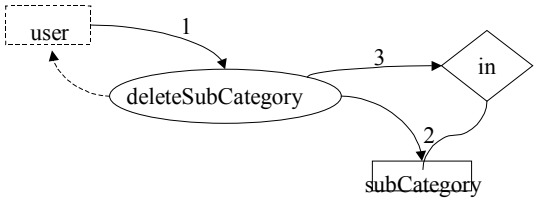
2: categoryCode

3: subCategoryCode, description

4: categoryCode

When a new sub category is added, the process checks that the input categoryCode is present in category.

deleteSubCategory



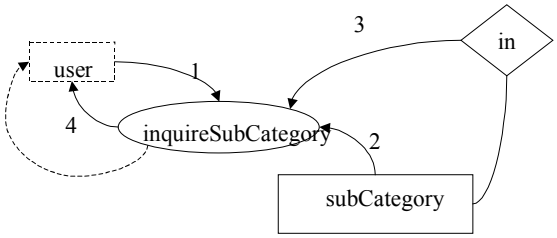
1: subCategoryCode

2: subCategoryCode, description

3: categoryCode

It returns an error in the case the sub category taken as input (described by subCategoryCode) does not exist.

inquireSubCategory



1: subCategoryCode

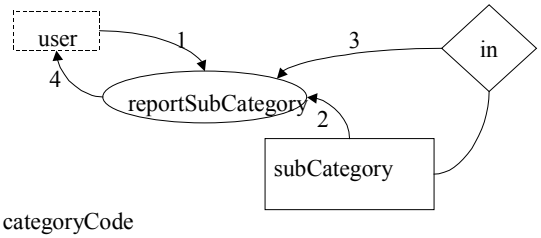
2: subCategoryCode, description

3: categoryCode

4: subCategoryCode, description, categoryCode

It returns an error in the case the sub category taken as input (described by subCategoryCode) does not exist.

reportSubCategory



1: categoryCode

2: subCategoryCode, description

3: categoryCode

4: subCategoryCode, description, categoryCode

References

[FGM88] Fuggetta, A., Ghezzi, C., Mandrioli, D. and Morzenti, A. (1988) *VLP: a Visual Language for Prototyping*. *IEEE Workshop on Languages for Automation*, College Park, MD, August.

[Che76] Chen, P. P. (1976) The Entity-Relationship model. Toward a unified view of data. *ACM Transactions On Database System*, **1(1)**.

[DeM78] DeMarco, T. (1978) *Structured Analysis and System Specification.* Yourdon Press, New York.

[GR02] Golfarelli, M., Rizzi, S., (2002) *Data Warehouse: Teoria e pratica della progettazione.* McGraw-Hill, Milano.

[LMR97] E. Lamma, P. Mello, F. Riguzzi, *A System for Measuring Function Points*, Technical Report DEIS-LIA-97-006.