Improving Use Case Based Requirements Using Formally Grounded Specifications *

C. Choppy and G. Reggio

[1] LIPN, Institut Galilée - Université Paris XIII, France
[2] DISI, Università di Genova, Italy

Abstract. Our approach aims at helping to produce adequate requirements, both clear and precise enough so as to provide a sound basis to the overall development. Our idea here is to find a way to combine both advantages of use cases and of formal specifications. We present a technique for improving use case based requirements, using the formally grounded development of the requirements specification, and that results both in an improved *requirements capture*, and in a *requirement validation*. The formally grounded requirements specification is written in a "visual" notation, using both diagrams and text, with a formal counterpart (written in the CASL and CASL-LTL languages). Being formally grounded, our method is systematic, and it yields further questions on the system that will be reflected in the improved use case descriptions. The resulting use case descriptions are of high quality, more systematic, more precise, and its corresponding formally grounded specification is available. We illustrate our approach on part of the Auction System case study.
KEYWORDS: requirements specification, specification methodology, formal specifications, CASL and CASL-LTL languages

1 Introduction

While tools and techniques are now available to support quite efficiently software development, one of the most difficult part remains to produce adequate requirements, both clear and precise enough so as to provide a sound basis to the overall development. Formal based specifications are advocated since they lead to precise, unambiguous descriptions, but they remain difficult to use and impractical in quite a number of cases. We think the reason for this is twofold. One point that was often put forward is the difficulty to write and read such specifications. Another point we see is that it may be difficult to start with formal specifications while still working on the requirements (thus, trying to understand what is the problem about).

Use cases were introduced [9] to capture the requirements of a system. They are quite successful because they easily give an idea of the system that can be discussed with the client. However, examples show that they are often imprecise, and also that the terms used are vague or ambiguous.

Our idea here is to find a way to combine both advantages of use cases and of formal specifications. We present a technique for improving use case based requirements, developing an associated Formally Grounded specification, and that results both in an improved *requirements capture* (some requirements may be updated and some may be

* Work supported by the Italian National Project SAHARA (Architetture software per infrastrutture di rete ad accesso eterogeneo).

new), and in a *requirement validation* since writing the specification leads to check that the requirements can be further made explicit up to a precise specification. The produced requirements specification is written in a "visual" notation, using both diagrams and text, with a formal counterpart which is written in the CASL[11] and CASL-LTL[12] specification languages. Being formally grounded, our method is systematic, and it yields further questions on the system that will be reflected in the improved use case descriptions. The resulting use case descriptions are of high quality, more systematic, more precise, and their corresponding Formally Grounded specification is available.

Use cases were introduced by Jacobson [9] after the earlier idea of scenarios, which are the different possible courses that different instances of the same use case can take. Use cases are used to describe/capture the requirements of software systems, while providing an overall picture of what is happening in the system. The use case description is textual (it should be "familiar", easy to read) and sums up a set of scenarios. Use cases should address the functional requirements, but it may be effective to adopt a *goal*-based approach, where a goal summarizes some functional requirements, thus providing a level of abstraction. A goal is an objective the system considered should achieve, and goals provide the rationale for requirements.
"A use case is a description of interactions between the system under discussion and external actors, related to the goal of one particular actor." [6].
Goal identification may not be easy since some goals may not be explicitely stated, thus requiring some elicitation [10], and there is often an intertwining of goal identification and requirement elaboration.

Use cases are popular because they are easy to use and informal, however "use cases are wonderful but confusing" [6]. A both good and bad thing is that there is a lot of freedom in what should include a use case description, and how it should be written. UML [17] proposes a diagram for use cases, states that descriptions are needed too, and that the sequence of use case activities are documented by behaviour specifications (e.g., with interaction diagrams).

Since use cases are written in the early phases of software development, it is crucial that they should be worked out with a lot of care, so as to avoid to generate errors that will be difficult and costly to correct further on. Interesting work is done to propose some guidelines on how to write use case descriptions, while keeping several levels obviously needed to cater for different stages and contexts, e.g., [6] distinguishes between "brief", "casual" and "fully-dressed" use cases, and proposes *templates* for structuring their descriptions. In the following, we use fully-dressed use cases, with an adaptation of this template provided by [13]. [6] also suggests some granularity levels in use cases with three levels, the *summary level*, the *user-goal level*, and the *subfunction level*, that we use too. The summary level use cases shows the sequencing of related goals, and provide a table of contents for lower level cases. A user-goal use case describes a goal of the primary actor, while subfunction level use cases are required to carry out user goals.

In Sect. 2 we shortly sum up our Formally Grounded approach, see [5] for a full presentation with other examples, for writing the requirements specification of a software system, in Sect. 3 we present our method for improving use case based requirements using Formally Grounded specifications, and in Sect. 4, we then show how our method applies to a part of the Auction System case study (the complete version is in [4]), showing how the starting use case based requirements have clarified, and how many

relevant aspects of the system have been enlightened, before concluding and discussing some related work in Sect. 5.

2 Our Formally Grounded Approach for Requirement Specification

Our Formally Grounded specification approach (see [5] for a complete presentation), aims at helping the user to understand the system to be developed, and to write the corresponding formal specifications. We also support visual presentations of formal specifications, so as to "make the best of both formal and informal worlds". We developed this method for the (logical-algebraic) specification language CASL [11] (Common Algebraic Specification Language, developed within the joint initiative CoFI[1]), and for an extension for dynamic systems CASL-LTL[2] [12]. Hence, for each visual specification, its formal counterpart in CASL or CASL-LTL is given.

In this approach, we follow the basic modelling ideas explained hereafter. We assume that the modelling entities are of three kinds, (i) a data structure (or data type), (ii) a simple dynamic system, that is a single dynamic entity, and (iii) a structured dynamic system, that is composed of mutually interacting dynamic entities.

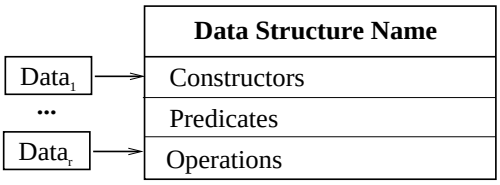
Each modelling entity considered may be modularly decomposed - so its (sub)*parts* are identified-, and is characterized by its *constituent features*. Their model/specification consists of a visual presentation of these parts and constituent features, and of their properties expressed in a natural-language style notation based on an appropriate underlying logic (the variant of logic depends on the kind of entity).

Once the constituent features are identified, we provide guidelines for an exhaustive search of the properties. To this end, we use a tableau whose cells, indexed by the pairs of constituent features, should be filled. For each cell we give a schemata for the relevant properties it should contain, expressing either, when the two indexes are different, the mutual relationships between the two features, or, when they are equal, what is known on that feature. This tableau ensures that no crucial part of the specification is forgotten. This technique results in producing a quite structured/navigable set of properties, which should be suitable to support evolution.

Our method caters for three different kinds of software items (a simple dynamic system, a structured dynamic system, or a data structure), while keeping a common "meta"-structure and way of thinking. We are thus providing support for the "building-bricks" tasks of specifying/modelling software artifacts that in our experience are needed for the development process.

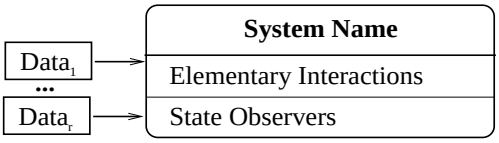
Data Structures Data structures are characterized by a set of values, some constructors to denote those values, and some predicates and operations. Data structures may be structured, e.g., they may import other data structures. These features and the imported data structures are visually presented as follows.

[1] `http://www.cofi.info`
[2] LTL stands for Labelled Transition Logic [7, 1].



Their properties are expressed in a many-sorted, first order logic [11] with a natural language-like notation. We provide a systematic technique to find the respective properties of constructors, predicates and operations, e.g., definedness, truth of predicate, etc. [5].

Simple Dynamic Systems Simple dynamic systems are characterized by their states and their transitions, where each transition corresponds to a change of state triggered by a set of elementary interactions with the external world. Each elementary interaction is described by a name and possibly by parameters (data values). The states are abstractly characterized by "state observers", which, given some parameter, may return some value (operations) or the truth of some condition (predicates). Thus, the constituent features of a simple (dynamic) system are its *elementary interactions* with the external environment and its *state observers*. The parts of a simple system are its *data structures* needed to define the parameters and the results of elementary interactions and state observers. The following *interface diagram* visually presents which are the subparts and the constituent features, while the specification of the parts is given separately.



The language we use for this description comprises both diagrams and text. These properties are expressed in LTL (Labelled Transition Logic) [7, 1], a branching time temporal (many-sorted, first-order with edge formulae) logic, with a natural language-like notation. This notation uses combinators for expressing that elementary interactions take place (e.g., e **happened**), standard logics (**if then else**, **not**, **and**, $=$, **exists**, ...), and also temporal combinators (e.g., **eventually**, **before**, **in any case**, **in one case**[3]). Both diagrams and text have a formal counterpart in the CASL-LTL language [12].

A transition from one state to another is characterized by elementary interactions, and properties about states and transitions are expressed, e.g., pre- and postconditions for elementary interactions, or incompatibilities between them. Properties for a state observer explore e.g., which elementary interactions may cause a change in its value, or which are its possible changes. Again here, we provide a systematic technique to find these properties that we recommend to express both in natural language and in our formal notation.

Structured Dynamic Systems A structured (dynamic) system is a specialized simple system which is composed of several dynamic systems, its *subsystems*, which can in turn be simple or structured. A transition of such a system should reflect which are

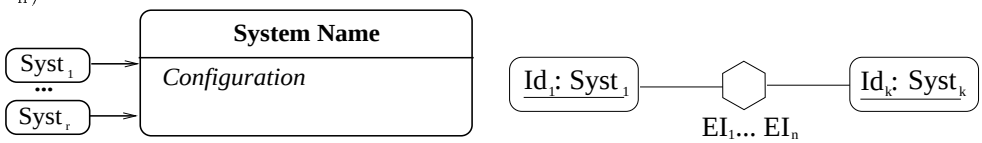
[3] The last two express universal and existential quantification over execution paths.

the subsystems transitions that occur. Moreover it is necessary to describe how the subsystems synchronize.

We present here a simpler version of structured systems (the general case is given in [5]) that have only simple subsystems (possibly of different types), where two subsystems may interact only pairwise by performing simultaneously the same elementary interaction, i.e., the behaviour of these structured systems is given by transitions made of groups of subsystem transitions, where each elementary interaction of a subsystem is matched by one of another subsystem. Moreover the considered structured systems are closed, i.e., they have no interaction with the external world.

A structured system is visually presented by

- (i) a *configuration diagram* showing the *subsystems*, and their types (the specifications of all those types are given separately),
- (ii) a *cooperation diagram* showing the *cooperations* among the subsystems (each cooperation is given by the synchronized execution of a an elementary interaction EI_n)



Configuration Diagram Cooperation Diagram

- (iii) a Data View which puts together all the specifications of *data structures* appearing in the specification of the system (parts of it and of its subsystems).

To specify the requirements on a software System it is sufficient to specify a structured dynamic system, whose subsystems are the System itself and all those entities interacting with it (context entities). The specification of the System will be the requirements, whereas the specifications of the context entities will show the assumptions made by the System on the context entities. The configuration and cooperation diagrams together show which is the context of the System, and so will be collected together in a Context View.

3 The Method for High-Quality Requirements

We present in this paper our method for producing enhanced requirements. It is organized in five tasks, and works on use case based requirements while developing a formally grounded specification, which results in improved requirements.

Task 1 Give the use case based requirements on the System following the method proposed by S. Sendall and A. Strohmeier in [14].

In general, in the given requirement specification there are several summary level use cases; below we present how to handle the first one; the remaining ones will be handled in the same way, the only difference is that instead of building a Formally Grounded specification from scratch, we enrich an existing one (developed for the first summary level use case considered), by adding more subsystems, constituent features and properties. The chosen summary level use case will be considered together with all its sub use cases at user and subfunction level. For each summary level use case together with all its included subcases do what follows.

Task 2 Find which are the external entities playing the roles corresponding to the *primary actors* (context entities) and determine their types. At this point we can draw a first version of the Context View, by depicting the System and the found context entities together with their types; the cooperation diagram will have only arcs connecting the System with the context entities.

Task 3 By examining the use case descriptions, one after the other, starting from the summary level one, look for *elementary interactions* and *state observers* of the System; the first should model the interactions between the System and the actors appearing in the use case scenarios, whereas the latter should model information recorded in the System examined or updated in the use case scenarios. They should be depicted in the *interface diagram* of the System specification (together with the type of their arguments and/or results); the elementary interactions should also be reported in the *cooperation diagram* to show which context entities are taking part in that interactions.

In the meantime put in the Data View any data structure that is used as an argument or a result by a found elementary interaction or state observer.

The association between use cases and the elementary interactions and state observers related to it (i.e., which are needed to describe it) should be recorded.

During this task, it is possible to find *new entities* interacting with the System that do not correspond to primary actors; they should be added to the Context View, together with their specifications.

Whenever, there are relevant assumptions on the context entities they should be made explicit by giving their Formally Grounded specification (they are just simple dynamic systems).

Task 4 Find all the properties about the System following our method. When filling a cell related to some constituent features (elementary interactions and state observers), the descriptions of the associated use cases should be examined as a source of inspiration. During this task, probably, *new state observers* and *data structures* will be added, and perhaps the *parameters* of the existing elementary interactions and state observers *may be modified.*

Task 5 During the tasks 3 and 4 many *questions* about the System will arise, many aspects of the System which need to be investigated will be highlighted, and many aspects of the System precisely described by the use cases will be found not convincing. These points may be settled following the usual ways, e.g., by interacting with the client, if available, by doing more investigation on the application domain, or by looking at existing similar systems. The produced Formally Grounded specification should reflect the System where all these points have been settled.

The original use case based requirement specification should then be *revised* so as to be coherent with the Formally Grounded one. In general use cases and/or scenarios may be added or removed, scenarios may be modified by adding/removing steps or by making more precise the terminology used to describe them.

In this way *the final outcome of our method will be not only a better and more systematic understanding of the System reflected in a Formally Grounded specification of the requirements, but also a more precise and sound use case based specification of the same requirements.*

Clearly task 5 will be performed in parallel with tasks 3 and 4.

4 The Auction System Case Study

In this paper we present the application of our method to the case study of an Auction System proposed in [13] by S. Sendall. The description of the problem solved by the Auction System is shown in Fig. 1.

Your team has been given the responsibility to develop an online auction system that allows people to negotiate over the buying and selling of goods in the form of English-style auctions (over the Internet). The company owners want to rival the Internet auctioning sites, such as, eBay, and uBid. The innovation with this system is that it guarantees bid placed are solvent, making for a more serious marketplace.
All potential users of the system must first enroll with the system; once enrolled they have to log on to the system for each session. Then, they are able to sell, buy, or browse the auctions available on the system. Customers have credit with the system that is used as security on each and every bid. Customers can increase their credit by asking the system to debit a certain amount from their credit card.
A customer that wishes to sell initiates an auction by informing the system of the goods to auction with the minimum bid price and reserve price for the goods, the start period of the auction, and the duration of the auction, e.g., 30 days. The seller has the right to cancel the auction as long as the auction's start date has not been passed, i.e., the auction has not already started.
Customers that wish to follow an auction must first join the auction. Note that it is only possible to join an active auction. Once a customer has joined the auction, (s)he may make a bid, or post a message on the auction's bulletin board (visible to the seller and all customers who are currently participants in the auction). A bid is valid if it is over the minimum bid increment (calculated purely on the amount of the previous high bid, e.g., 50 cent increments when bid is between $1-10, $1 increment between $10-50, etc.), and if the bidder has sufficient funds, i.e., the customer's credit with the system is at least as high as the sum of all pending bids. Bidders are allowed to place their bids until the auction closes, and place bids across as many auctions as they please. Once an auction closes, the system calculates whether the highest bid meets the reserve price given by the seller (English-style auction reserve price), and if so, the system deposits the highest bid price minus the commission taken for the auction service into the credit of the seller (credit internal with the system).
The auction system is highly concurrent–clients bidding against each other in parallel, and a client placing bids at different auctions and increasing his/her credit in parallel.

Fig. 1. Auction System Problem Description

4.1 Auction System Task 1 – Use Case Based Requirement Specification

Here we report the use case based specification of the requirement on the Auction System given following the method of [14] as found in [13]. The only difference with [13] is that we summarize the actors and the use cases by means of a UML use case diagram, see Fig. 2, below, showing also the "*include*" relationships among the use cases (depicted by dotted lines). In the following use case descriptions "**" means that the details about some aspects of the Auction System (e.g., data format or rules to

follow to perform some activity) are given in an accompanying document, not given in [13] and thus not considered here.[4].

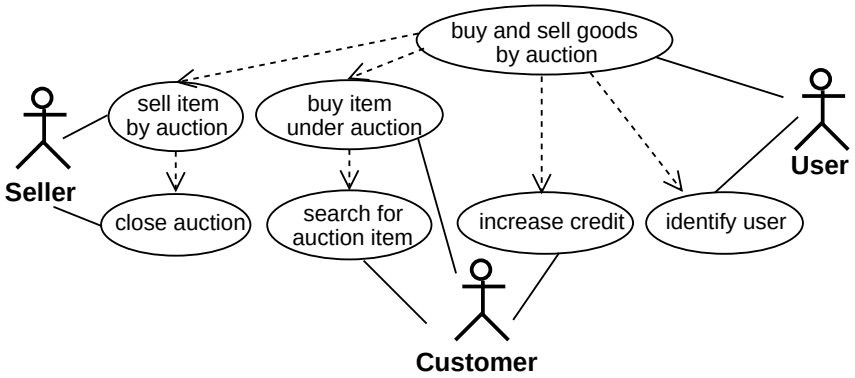


Fig. 2. Auction System: Use cases and actors

Here we do not detail the schema for the use case description followed in this example and presented in [14]. As introduced in Sec. 1, ue cases may be of different granularity levels, summary, user and subfunction. We present only the main use case buy item under auction, (which is of user level), the other ones are in the appendix Sect. A.

Use Case buy item under auction

Level: User Goal
Intention in Context: The intention of the Customer is to follow the auction, which may then evolve into an intention to buy an item by auction, i.e., (s)he may then choose to bid for an item. The Customer may bid in many different auctions at any one time. (Also the actor Participant means the Seller and all the Customers that are joined to the auction).
Primary Actor: Customer
Precondition: The Customer has already identified him/herself to the System.
Main Success Scenario:
Customer may leave the auction and come back again later to look at the progress of the auction, without effect on the auction; in this case, the Customer is required to join the auction again.
1. Customer searches for an item under auction (search item).
2. Customer requests System to join the auction of the item.
3. System presents a view of the auction** to Customer.

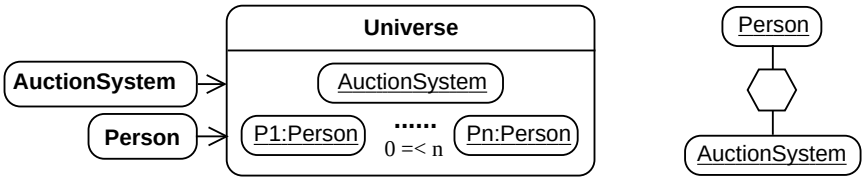
Steps 4-5 can be repeated according to the intentions and bidding policy of the Customer
4. Customer makes a bid on the item to System.
5. System validates the bid, records it, secures the bid amount from Customer's credit**, releases the security on the previous high bidder's credit (only when there was a previous standing bid), informs Participants of new high bid, and updates the view of the auction for the item** with new high bid to all Customers that are joined to the auction.

[4] However, if such accompanying documents were available, they might be considered to complete the Formally Grounded specification, and they might undergo a similar revision process.

Customer has the high bid for the auction.
6. System closes the auction with a winning bid by Customer.
Extensions:
2a. Customer requests System not to pursue item further:
2a.1. System permits Customer to choose another auction, or go back to an earlier point in the selection process; use case continues at step 2.
3a. System informs Customer that auction has not started: use case ends in failure.
3b. System informs Customer that auction is closed: use case ends in failure.
4a. Customer leaves auction:
4a.1a. System ascertains that Customer has high bid in auction:
4a.1a.1. System continues auction without effect; use case continues at step 6
4a.1b. System ascertains that Customer does not have high bid in auction: use case ends in failure.
4||a. Customer requests System to post a message to auction and provides the message content**.
4||a.1. System informs all Participants of message; use case continues from where it was interrupted.
5a. System determines that bid does not meet the minimum increment**:
5a.1. System informs Customer; use cases continues at step 4.
5b. System determines that Customer does not have sufficient credit to guarantee bid:
5b.1. System informs Customer; use cases continues at step 4.
6a. Customer was not the highest bidder:
6a.1. System closes the auction; use case ends in failure.

4.2 Auction System: Task 2

The Auction System has any number of context entities all of the type Person (anyone accessing the system by Internet). A Person may play three roles: User (plain Internet user), Customer (a User identified by the Auction System and connected with it) and Seller (a Customer selling some goods using the Auction System). We give a first version of the Context View showing the Auction System and the Persons (the incomplete cooperation diagram just shows that the Persons interact only with the Auction System).



4.3 Auction System: Task 3

We examine the various use cases, one after the other, starting from the summary level one, followed by the included user goal and subfunction ones, looking for the elementary actions and the state observers of AuctionSystem, together with the needed data structures and, possibly new context entities. Note that, for each use case, we do not give the features used by the included sub-use cases.

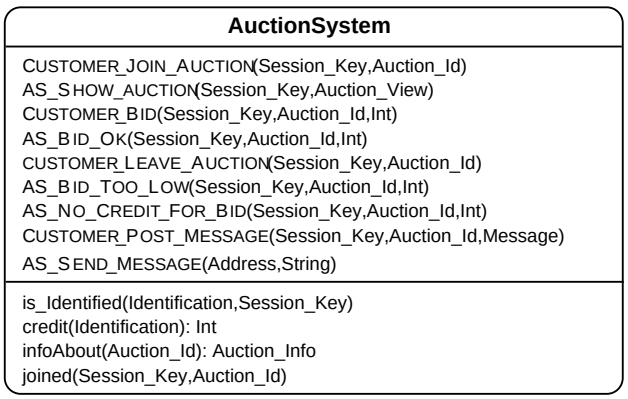
We name each elementary interaction made by the Auction System with an identifier of the form AS_..., whereas those made by a person context entity will be named User_..., Customer_... and Seller_..., depending on the role.

For each use case we produce a fragment of the Context View, of the Data View and of the interface of the specification of the Auction System. At the end, all these fragments will be put together getting the initial view of the structural part of the Formally Grounded requirement specification of the Auction System. To be able to support the evolution of the requirements, however, we require to keep track of the features of the Formally Grounded specification (elementary interactions, state observers, and data structures) that are related with each use case. Here, we just add comments lines in the various diagrams.

Already, during this task many questions about the Auction System may arise that should be settled with the client; we use the following annotation for these questions and the way chosen to settle them **Q:** *problem* **A:** *settled in this way.*

Here we show how the detail about one use case (buy item under auction); in Appendix A, we present instead the final outcome of this task (Data View, Context View and the interface of the Auction System) and the many problems about the Auction System found while performing it.

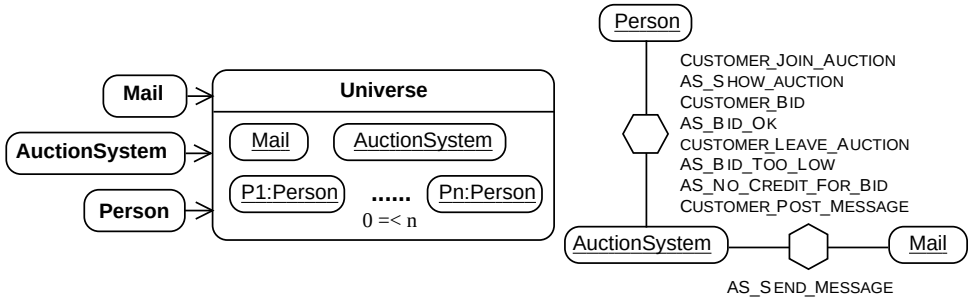
Use Case buy item under auction



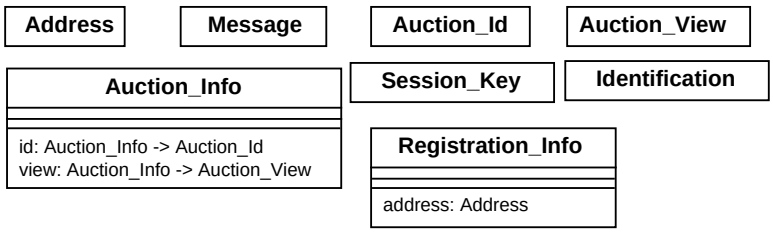
The elementary interactions of AuctionSystem, shown in the first compartment of the above interface diagram, correspond either to an interaction made in the use case by the Auction System towards a context entity (e.g., AS_BID_OK for communicating that the placed bid was ok) or to an interaction received by a context entity (e.g., CUSTOMER_BID for a Customer placing a bid). Instead, the state observers, in the second compartment, correspond to information recorded inside the Auction System either tested or updated during the use case (e.g., *credit*: the actual credit of a Customer denoted by an identification; *infoAbout*: the current information about an auction).

Context View **Q:** *This use case requires that the Auction System informs the participants to an auction about various facts (for example, when there is a new higher bid or a new message of another participant), but nothing is said on how that will be performed. In the description of the use case* close auction *there is a note saying that this is an open issue and that it will be likely made by email.* **A:** *We settle this point by assuming the existence of an external mail service, not further detailed, able to deliver messages to User identified by some kind of address. The mail service will be then a NEW context entity (and a new secondary actor).*

The context view shows which context entities take part in the use case (e.g., the mail service), and which are the interactions of the Auction System with them (e.g., AS_SEND_MESSAGE is an interaction between AuctionSystem and the mail service).



Data View It shows all the data used as parameters by the found elementary interactions and state observers, and which predicates/operations we surely need to perform all the calculations over them required by the use case. For example, **Auction_Info**, the information about an auction, has an operation, *view*, for recovering a view of the auction to be shown to its participants, whereas **Auction_View** is not further detailed.



4.4 Auction System: Task 4

This task consists in finding the properties about the Auction System by filling the tableaux generated by the elementary interactions and state observers found in the previous task, and by completing the specifications of the data structures. Clearly, while doing this activity, new state observers may be introduced, which will have then to be introduced in the tableaux and considered while looking for the properties. The original use case based specification may be modified by reflecting the better insights on the Auction System gained while looking for properties.

Here we show only some properties about a few elementary interactions and state observers needed for the use case buy item under auction; each property is both expressed in our notation, and accompanied by a comment. The full set of the properties can be found in the appendix Sect. A.

Elementary Interaction CUSTOMER_JOIN_AUCTION

Looking for the precondition of CUSTOMER_JOIN_AUCTION we found the following unclear points about the Auction System.

Q: *Does the use case search item ends having selected one auction or one item? This is relevant because there may be many different auctions for the same item, e.g., a used car.* **A:** *The description of search item suggests some auctions, whereas that of buy item*

under auction suggests one item. We assume that search item *ends with some selected auctions.*
Q: *Can the auction selected by the* search item *be in any status (e.g., closed or not yet started)?* **A:** *Yes, and this is quite sensible, since a Customer may be interested in knowing that some item has been sold in the past and at which price, or which are the current starting prices of some items, or that some items will be soon auctioned.*
Q: *Can a Customer try to join a closed or not-started auction?* **A:** *No, the Auction System should not provide this possibility, instead of answering with an error.*

The above problems lead us to *revise* the use case search item by discussing with the client. As a result, we now have the user goal level browse auctions use case ending with a selected group of auctions (indeed a Customer may be interested just in browsing all auctions handled by the Auction System). Moreover, the use case buy item under auction may start only when there is one selected auction, which is active. Then, we introduce a new state observer *selected_Auctions* that associates with each identified Customer (referred by a session key) the identities of the currently selected auctions.
Q: *Can a Customer join an auction to which (s)he is already joined?* **A:** *Yes, since there is no problem, also if a better choice may be that the Auction System send a warning to Customer.*

If a Customer joins an auction, then
  (s)he is identified,
  Customer has selected exactly one auction, which is active;
  and after (the Customer has joined that auction, and
  the Auction System shows to her/him all the detail of the selected auction)
if CUSTOMER_JOIN_AUCTION(*sk*) **happen then**
  exists *id*:*Identification* **s.t.** *is_Identified*(*id*,*sk*) **and**
  exists *aid*:*Auction_Id* **s.t.**
    $selected\_Auctions(sk) = \{aid\}$ **and** $status(infoAbout(aid)) = active$ **and**
    $joined(sk, infoAbout^{nxt}(aid))$ **and**
    in any case next AS_SHOW_AUCTION(*sk*,*view*(*infoAbout*(*aid*))) **happen**

Elementary Interaction AS_BID_OK

While looking for the postcondition of AS_BID_OK, which may be also about the future behaviour of the Auction System after having performed the elementary interaction, we detected the following problem.
Q: *Is true that a Customer joined to an auction is informed twice of each new bid, once by receiving a view of the auction with the new bid and once by some kind of message? Moreover, if a Customer places a bid, and after leaves the auction, will be her/him ever informed of a new higher bid? More generally, which is the intended duration of an auction? a few hours when the participants bid many times, and continuously look at the current view of the auction? or several days, when the participants from time to time place their bids and look at the situation of the auction?* **A:** *An auction handled by the Auction System should last a few hours with all participants logged on; thus there is no need to inform the joined customers and the seller of the various bids, because they continuously examine the current view of the auction that the Auction System keeps updated.*

If the Auction System informs a Customer that her/his bid is ok, then
  the Customer has placed such bid,

(s)he had sufficient credit, and the bid met the minimum increment; and after
the bid is recorded,
the amount is secured by the Customer credit,
the security on the previous high bid is released (if any), and
the updated auction view is sent to all the Customers joined to the auction.

if AS_Bid_Ok(sk, aid, i) **happen then**
in any case before Customer_Bid(sk, aid, i) **happened and**
$i \leq credit(identityOf(sk))$ **and**
$bid\_Ok(infoAbout(aid), i)$ **and**
$high\_Bidder(infoAbout^{nxt}(aid)) = identityOf(sk)$ **and** $high\_Bid(infoAbout^{nxt}(aid)) = i$ **and**
$credit^{nxt}(identityOf(sk)) = credit(identityOf(sk))$ - i **and**
(**if is defined** $(high\_Bidder(infoAbout(aid)))$) **then**
$credit^{nxt}(high\_Bidder(infoAbout(aid))) =$
$credit(high\_Bidder(infoAbout(aid))) + high\_Bid(infoAbout(aid)))$ **and**
for all sk_1:*Session_Key* •
if $infoAbout(sk_1, infoAbout(aid))$ **then** AS_Show_Auction($sk_1, view(infoAbout(aid))$)

State Observer *credit* The first version of the property about the decreasing of the credit based on what is written in the various use case descriptions is the following, and points out a problem.

The Auction System does not allow a Customer to be in debit, thus
$credit(id) \geq 0$
If the credit of a Customer increases, then
either the Customer asked the Auction System to increase it,
or a bid of the Customer has been overcame,
or an auction where the Customer is the seller ended successfully.
if $credit^{nxt}(id) = credit(id) + i$ **and** $i > 0$ **then**
exists *sk*, *ctd* **s.t.** $is\_Identified(id, sk)$ **and** $i = amount(ctd)$ **and**
AS_Increased_Credit(sk, ctd) **happen**
or
exists *aid* **s.t.** $high\_Bidder(infoAbout(aid)) = id$ **and**
$high\_Bidder(infoAbout^{nxt}(aid)) \neq id$ **and** $high\_Bid(infoAbout(aid)) = i$
or
exists *aid* **s.t.** AS_Send_Message($addressOf(id)$,"closed auction", aid, i) **happen**

If the credit of a Customer decreases, then
the Customer made a bid in an auction.
if $credit^{nxt}(id) = credit(id)$ - i **and** $i > 0$ **then exists** *sk*:*Session_Key*, *aid*:*Auction_Id* **s.t.**
AS_Bid_Ok(sk, aid, i) **happened and** $is\_Identified(id, sk)$

Q: *It is true that a Customer using the Auction System only for selling items will be never able to collect her/his money? Moreover, can a buying Customer recover her/his money when (s)he is no more interested in buying?* **A:** *Yes; thus we have to add a NEW use case* decrease credit *for allowing a Customer to recover her/his credit.*

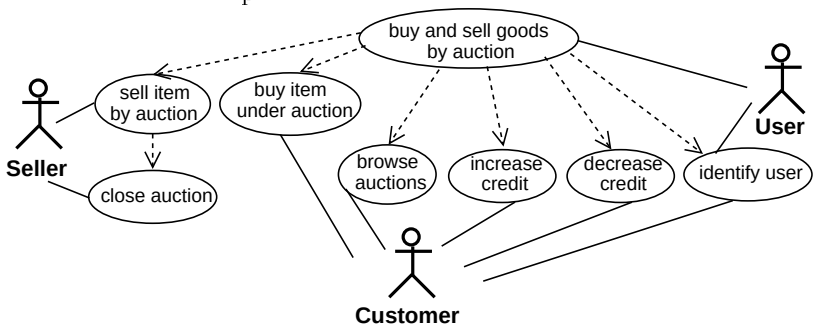
The new version we propose is then

If the credit of a Customer decreases, then
either the Customer asked the Auction System to decrease it, *(NEW)*
or the Customer made a bid in an auction.
if $credit^{nxt}(id) = credit(id)$ - i **and** $i > 0$ **then**
exists *sk*:*Session_Key*, *ctdCredit_Transfer_Detail* **s.t.**

AS_Decreased_Credit(sk,ctd) **happened and**
i= $amount(ctd)$ **and** $is\_Identified(id,sk)$
or exists sk:$Session\_Key$, aid:$Auction\_Id$ **s.t.**
AS_Bid_Ok(sk,aid,i) **happened and** $is\_Identified(id,sk)$

4.5 Auction System Task 5 – New Use Case Based Requirement Specification

Here we report only the new use case diagram and the new description of the use case buy item under auction, see [4] for the complete new use case based requirements. Two new use cases were identified when following our approach (see above), browse auctions (thus, point 1. was removed from the buy item under auction description below) and decrease credit. The questions brought up by our work led to several modifications, e.g., the work on AS_Bid_Ok in Sect. 4.4 led to remove one part of point 5. in the new buy item under auction description below.



Use Case buy item under auction

Level: User Goal
Intention in Context: *UNCHANGED*
Primary Actor: Customer
Precondition: The Customer has already identified him/herself to the System
NEW: and selected one active auction.
Main Success Scenario:
Customer may leave the auction and come back again later to look at the progress of the auction, without effect on the auction; in this case, the Customer is required to join the auction again.
REMOVED: 1. Customer searches for an item under auction (search item).
2. Customer requests System to join the selected auction.
3. System presents a view of the auction** to Customer.
Steps 4-5 can be repeated according to the intentions and bidding policy of the Customer
4. Customer makes a bid on the item to System.
5. System validates the bid, records it, secures the bid amount from Customer's credit**, releases the security on the previous high bidder's credit (only when there was a previous standing bid), *(REMOVED: informs Participants of new high bid,)* and updates the view of the auction for the item** with new high bid to all Customers that are joined to the auction.
Customer has the high bid for the auction
6. System closes the auction with a winning bid by Customer.
Extensions:

2a. Customer requests System not to pursue item further:
2a.1. System permits Customer to choose another auction, or go back to an earlier point in the selection process; use case continues at step 2.
3a. *NEW: The Customer is the Seller of the auction; System informs Customer that (s)he cannot join the auction. Use case ends with failure.*
REMOVED: 3a. System informs Customer that auction has not started: use case ends in failure.
REMOVED: 3b. System informs Customer that auction is closed: use case ends in failure.
4a. Customer leaves auction:
4a.1a. System ascertains that Customer has high bid in auction:
4a.1a.1. System continues auction without effect; use case continues at step 5
4a.1b. System ascertains that Customer does not have high bid in auction: use case ends in failure.
4||a. Customer requests System to post a message to auction and provides the message content**.
4||a.1. *MODIFIED: System updates the view of the auction with the added message to all Customers that are joined to the auction;* use cases continues from where it was interrupted.
UNCHANGED: 5a, 5b, 6a

5 Conclusion and related works

In this paper we have proposed a method to review use case based requirements for a system by building an accompanying Formally Grounded specification. As a result the initial requirements are examined in a systematic way by looking at the various aspects of the considered system, modelleld in terms of elementary interactions and state observers. For example, the possible interferences among different use cases may be revealed (elementary interactions relative to different use cases may update the same state observer), the communications between the system and the actors become more precise (they are modelled by elementary interactions, which require a precise definition of their parameters), the secondary actors (those helping the system to satisfy the goal of the primary actors) are discovered and their features made explicit (all entities interacting with the system must be defined and modelled).

The produced Formally Grounded specification has a user-friendly notation (diagrams plus textual annotations in a natural-like language), and so it could be used as the requirement document. However, the proposed method requires also to update the original use case based requirements whenever an aspect of the system is enlightened, thus, at the end new improved use case based requirements are available. In the meantime, the formal CASL/CASL-LTL specification corresponding to the Formally Grounded one is also available, e.g., for formal analysis.

We think that starting to build directly the Formally Grounded specification from the description of the problem may be not as much as effective as the proposed combination of use case and Formally Grounded specification, because the ingredients of the Formally Grounded specification (elementary interactions and state observers) are in some sense at a finer grain than the functionalities of the system, and so may be difficult to find by just considering the problem.

As an example, we have used our method on a medium-size case study: an electronic auction system, however, for lack of room, here we have described only parts of the various tasks and shown only some fragments of the produced artifacts; the complete development and the resulting artifacts can be found in [4]. The advantages

shown by our method on this case study seem quite positive. Indeed, we have detected many problematic or not completely clarified aspects in the original use case based requirements. Among them, we recall

- - explicit auctions browsing functionality (blurred in the initial requirements: the information on all auctions were available but not shown),
- the fact that the auctions should be performed in a chat-like way;
- discovered the need for a decrease-credit functionality;
- clarified the fact that someone may register under many different identities, so that two different Customers may be the same person;
- a Customer may disconnected by the System by hers/his own choice, and not only after sometime (s)he is doing nothing.
- a Customer cannot unregister from the System when (s)he is the seller or has the high bid in an auction.
- made explicit that when a Customer unregisters any left credit is seized by the Auction System owner;
- ...

Moreover, we would like point out, that the starting use case requirements [13] was not produced by ourselves, but instead by people without any relationship with our group and our method, and that they are quite accurate, presented using a well-organized template and produced following a good method.

Concerning the possibility to effectively using the proposed method we have the following positive points

- It is possible, using common existing technologies, to build software tools to support the construction of the Formally Grounded specification, not only a graphical editor, but also wizards guiding to find the properties;
- Each use case is linked with the elementary interactions, the state observers and the data structures used for its specification. This, together with the precise structure of the properties, may also help to support the evolution of the initial requirements; indeed a modification in one use case may be only reflected in a precise part of the associated Formally Grounded specification.
- The inspection and revision of the requirements proposed by our method concerns only the nature of the system to be developed, and does not require to make any choice about the technology and methods that will be used to realize the system; thus it may be used in combination with many different methods.

On the negative side, we have to recall that the method that we propose requires the additional task of building the Formally Grounded specification. We think that the developer, before to use it, should balance the following two needs

1. to correctly capture the functional aspects of the system before starting the development,
2. to have a working "version" of the system as soon as possible.

Obviously, the choice of which is more important between 1 and 2 depends on the nature of the project; for example, only few persons available only for a short period of time have the knowhow to give the requirements (+1), if the "client" is always available at the development team (+2), or the client has very few ideas about the system but

(s)he hopes that when (s)he sees what can be done by using an initial version of the system his/her ideas will clarify (+2).

In the literature there are other approaches trying to build a formal specification of the requirement of a system, but in general they do not aim at producing an improved non-formal specification. Among them we may recall the nice work of A. van Lamsweerde and his group (see, e.g., [18]), which offers a way to formally specify goal-oriented requirement specifications, and then to analyze them by means of formal techniques. Instead R. Dromey (see, e.g., [8]) proposes to use "Behaviour Tree" a formal-visual notation to specify the requirements, and then the resulting requirement specification will be used to derive the architectural structuring of the system. Instead our approach, in the line of the well-founded methods [3], uses the underlying formal foundation to get a rigorous method to precisely specify the requirements, with the aim of achieving a careful inspection and a kind of validation of those requirements.

One of the authors together with E. Astesiano have proposed another use case based method for the precise specification of the requirements [2], but using the (non-formal) UML statecharts as a notation to describe the use cases. However, because it does not offer a systematic way to analysis the System under different viewpoints, some aspects of the System captured by our method may not come under the light.

We would like also to quote the work by S. Sendall and A. Strohmeier [15, 16] who promote the use of operation schemas (pre- and postconditions written in OCL [19]) and system interface protocols (describing the temporal ordering of system operations with UML state diagrams) to complement use cases, and to introduce timing constraints. Our goal is different, that is to improve the use case based requirements, and our approach is also specific since we show the questions raised by our process, and how the answers lead to the improvements. Another important point is that our companion method for formally grounded specification provides a systematic search for relevant properties. Finally, the corresponding formal specification is structured and covers all aspects of dynamic complex systems.

References

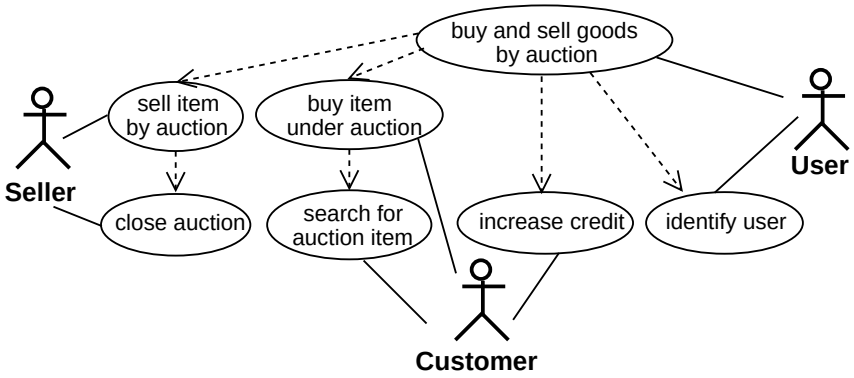
1. E. Astesiano and G. Reggio. Labelled Transition Logic: An Outline. *Acta Informatica*, 37(11-12), 2001.
2. E. Astesiano and G. Reggio. Tight Structuring for Precise UML-based Requirement Specifications. In *Proc. of Monterey Workshop 2002: Radical Innovations of Software and Systems Engineering in the Future. Venice - Italy, October 7-11, 2002.*, Lecture Notes in Computer Science. Springer Verlag, Berlin, 2003. To appear. Available at `ftp://ftp.disi.unige.it/person/ReggioG/AstesianoEtAll03f.pdf`.
3. E. Astesiano, G. Reggio, and M. Cerioli. From Formal Techniques to Well-Founded Software Development Methods. In *Proc. of The 10th Anniversary Colloquium of the United Nations University International Institute for Software Technology (UNU/IIST): Formal Methods at the Crossroads from Panacea to Foundational Support. Lisbon - Portugal, March 18-21, 2002.*, Lecture Notes in Computer Science. Springer Verlag, Berlin, 2003. To appear. Available at `ftp://ftp.disi.unige.it/person/ReggioG/AstesianoEtAll03a.ps` and `ftp://ftp.disi.unige.it/person/ReggioG/AstesianoEtAll03a.pdf`.
4. C. Choppy and G. Reggio. Improving Use Case Based Requirements Using Formally Grounded Specifications (Complete Version). Technical Report DISI-TR-03-45, DISI – Università di Genova, Italy, 2003. Avail-

able at ftp://ftp.disi.unige.it/person/ReggioG/ChoppyReggio03c.ps and ftp://ftp.disi.unige.it/person/ReggioG/ChoppyReggio03c.pdf.
5. C. Choppy and G. Reggio. Towards a Formally Grounded Software Development Method. Technical Report DISI–TR–03–35, DISI, Università di Genova, Italy, 2003. Available at ftp://ftp.disi.unige.it/person/ReggioG/ChoppyReggio03a.pdf.
6. A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000.
7. G. Costa and G. Reggio. Specification of Abstract Dynamic Data Types: A Temporal Logic Approach. *T.C.S.*, 173(2), 1997.
8. R. Dromey. From Requirements to Design: Formalizing the Key Steps. In *Proc. of SEFM'03, Brisbane - Australia*. IEEE Computer Society, 2003.
9. I. Jacobson, M. Christerson, P. Jonnson, and G. Overgaard. *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Addison-Wesley, 1992.
10. A. V. Lansweerde. Goal-Oriented Requirements Engineering: A Guided Tour (Invited Lecture). In *Proc. RE'01 - 5th International Symposium on Requirements Engineering*, Toronto, 2001.
11. P. Mosses, editor. *CASL, The Common Algebraic Specification Language - Reference Manual*. Lecture Notes in Computer Science. Springer-Verlag, 2003. To appear. Available at http://www.cofi.info/CASL_RefManual_DRAFT.pdf.
12. G. Reggio, E. Astesiano, and C. Choppy. CASL-LTL: A CASL extension for dynamic reactive systems – summary. Technical Report DISI-TR-99-34, Università di Genova, Italy, 1999. Revised February 2000, August 2003.
13. S. Sendall. Case studies for RE_A2 course "Requirements Analysis with Use Cases". http://lglwww.epfl.ch/research/use_cases/RE-A2-case-studies/index.html, 2001.
14. S. Sendall and A.Strohmeier. Requirements Analysis with Use Cases. http://lglwww.epfl.ch/research/use_cases/RE-A2-theory.pdf, 2001.
15. S.Sendall and A.Strohmeier. From Use Cases to System Operation Specifications. In S. K. A. Evans and B. Selic, editors, *Proc. UML'2000*, number 1939 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 2000.
16. S.Sendall and A.Strohmeier. Specifying Concurrent System Behavior and Timing constraints using OCL and UML. In M. Gogolla and C. Kobryn, editors, *Proc. UML'2001*, number 2185 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 2001.
17. UML Revision Task Force. *OMG UML Specification 1.3*, 2000. Available at http://www.omg.org/docs/formal/00-03-01.pdf.
18. A. van Lamsweerde. Building Formal Requirements Models for Reliable Software (Invited paper). In *6th International Conference on Reliable Software Technologies, Ada-Europe 2001*, number 2043 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 2001.
19. J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1999.

A Auction System: the complete development of the case study

A.1 Auction System Task 1 – Use Case Based Requirement Specification

Here we report the complete initial use case based specification of the requirement on the Auction System of S. Sendall [13] given in Fig. 1. We provide again the use case diagram given in Fig. 2.



The use cases to be considered are: buy and sell goods by auction, identify user, increase credit, buy item under auction, sell item by auction, close auction, and search item.

Use Case buy and sell goods by auction

Level: Summary
Intention in Context: The intention of the User is to buy and sell goods by auctions over time. A User can be involved in multiple auctions at any one time.
Primary Actor: User (becomes Customer once (s)he has identified him/herself with the System)
Main Success Scenario:
All Users must first enrol with the System before they have the right to use it
1. User enrols with System, providing System with registration information**.
2. System validates the registration information and enrols the User.
Steps 3-7 can be repeated according to the intention of the User to buy, sell and brows over time
3. User identifies him/herself to System (identify user).
Steps 4-6 can be performed in parallel and individually repeated.
A customer may bid and sell in many auctions at any one time.
4. Customer increases credit with System (increase credit).
5. Customer buys and sells items (buy item under auction and sell item by auction).
6. Customer exits System.
Extensions:
2a. System ascertains that User did not provide sufficient information to register him/her:
  2a.1. System informs User; use case continues at step 1.
3a. System fails to identify User:
  3a.1. System prompts User to enrol; use case continues at step 1.
(4-5)||a. System detects that Customer has exited the System (defined by some sort of time-out on user response (open issue): use case continues at step 3.

Use Case identify user

Level: Subfunction
Intention in Context: The intention of the User is to identify him/herself so that (s)he may participate in auction(s).
Primary Actor: User
Main Success Scenario:
1. User identifies him/herself with System by providing identification**.
2. System validates the identification.
Extensions:
2a. System fails to identify User:
  2a.1. System informs User; use case ends in failure.

Use Case increase credit

Level: User Goal
Intention in Context: The intention of the User is to increase his/her credit with the System so that (s)he may buy goods on auction(s).
Primary Actor: Customer
Precondition: The Customer has already identified him/herself to the System.
Main Success Scenario:
1. Customer requests System to increase his/her credit, and provides credit transfer details**.
2. System validates the transfer details.
3. System transfers credit from the Customer's Credit Organization and informs Customer of a successful outcome.
Extensions:
2a. System ascertains that insufficient information was provided:
  2a.1. System informs User; use case goes back to step 1.
2b. System ascertains that incorrect information was provided:
  2b.1. System informs User; use case goes back to step 1.
3a. System is unsuccessful with the credit transfer:
  3a.1. System informs User; use case ends in failure.

Use Case buy item under auction

Level: User Goal
Intention in Context: The intention of the Customer is to follow the auction, which may then evolve into an intention to buy an item by auction, i.e., (s)he may then choose to bid for an item. The Customer may bid in many different auctions at any one time. (Also the actor Participant means the Seller and all the Customers that are joined to the auction).
Primary Actor: Customer
Precondition: The Customer has already identified him/herself to the System.
Main Success Scenario:
Customer may leave the auction and come back again later to look at the progress of the auction, without effect on the auction; in this case, the Customer is required to join the auction again.
1. Customer searches for an item under auction (search item).
2. Customer requests System to join the auction of the item.
3. System presents a view of the auction** to Customer.
Steps 4-5 can be repeated according to the intentions and bidding policy of the Customer
4. Customer makes a bid on the item to System.
5. System validates the bid, records it, secures the bid amount from Customer's credit**, releases the security on the previous high bidder's credit (only when there was a previous

standing bid), informs Participants of new high bid, and updates the view of the auction for the item** with new high bid to all Customers that are joined to the auction. Customer has the high bid for the auction.
6. System closes the auction with a winning bid by Customer.

Extensions:

2a. Customer requests System not to pursue item further:
2a.1. System permits Customer to choose another auction, or go back to an earlier point in the selection process; use case continues at step 2.
3a. System informs Customer that auction has not started: use case ends in failure.
3b. System informs Customer that auction is closed: use case ends in failure.
4a. Customer leaves auction:
4a.1a. System ascertains that Customer has high bid in auction:
4a.1a.1. System continues auction without effect; use case continues at step 6
4a.1b. System ascertains that Customer does not have high bid in auction: use case ends in failure.
4||a. Customer requests System to post a message to auction and provides the message content**.
4||a.1. System informs all Participants of message; use case continues from where it was interrupted.
5a. System determines that bid does not meet the minimum increment**:
5a.1. System informs Customer; use cases continues at step 4.
5b. System determines that Customer does not have sufficient credit to guarantee bid:
5b.1. System informs Customer; use cases continues at step 4.
6a. Customer was not the highest bidder:
6a.1. System closes the auction; use case ends in failure.

Use Case sell item by auction

Level: User Goal
Intention in Context: The intention of the Seller is to sell an item by auction. A Seller may be the seller in many auctions at any one time. (Participants include the Seller and all the Customers that are joined to the auction)
Primary Actor: Seller
Precondition: The Seller has already identified him/herself to the System.
Main Success Scenario: Seller may leave the auction and come back again later to look at its progress, without effect on the auction.
1. Seller registers an item to sell on auction with System, providing registration information**.
2. System validates the information, presents a view of auction** to Seller, and records and publishes the new auction.
3. System closes auction (close auction) with a successful sale.

Extensions:

2a. System ascertains that it was not given sufficient information to define an auction:
2a.1. System informs Seller to provide more information; use case continues at step 1.
2b. System ascertains that it was not given acceptable information to define an auction:
2b.1. System informs Seller that certain details are not acceptable; use case continues at step 1.
(2-3)||b. Seller requests System to post a message to auction and provides the message content*.
(2-3)||b.1. System informs all Participants of message;
use case continues from where it was interrupted.
3a. System closes auction without a successful sale: use case ends in failure.
3b. Seller requests System to cancel auction:
3b.1a. System detects that auction has already started:

3b.1a.1. System refuses**; use case continues from where it was interrupted.
3b.1b. System detects that auction has not yet started:
3b.1b.1. System cancels and removes auction, and publishes the fact that the auction is canceled; use case ends in failure.

Use Case close auction

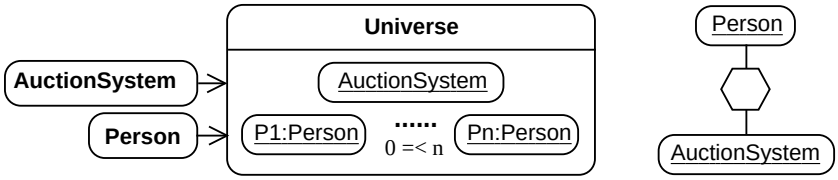
Level: Subfunction
Intention in Context: The Seller proposed an expiration date for the auction that the System (on behalf of the Auction System Owners) agreed to**. The System closes down the auction at this moment.
Primary Actor: (Seller)
Trigger: The System detects that the auction has expired.
Main Success Scenario:
1. System detects that auction has expired.
2. System validates that the highest bid has met Seller's reserve price.
3. The System closes down auction:
removes the amount of the winning bid [security] from the Customer's credit by commiting the withdrawal of the bid amount,
augments the Seller's source of credit by the winning bid amount minus the commission taken for the auction service,
informs the Customer with the winning bid that his/her bid was successful (open issue: this will most likely be done via email),
informs all participants in the auction of the result, and
informs the Seller of the result** and requests him/her to organize the delivery of the item with the Customer.
Extensions:
2a. System detects that highest bid has not met Seller's reserve price for the item:
2a.1. System releases the security from the credit of the Customer with the high bid, and it informs the participants and the Seller that there was no sale; use case ends in failure.
2b. System detects that no bid was made on auction:
2b.1. System informs the participants and the Seller that there was no sale;
use case ends in failure.

Use Case search item

Level: Subfunction
Intention in Context: The intention of the Customer is to find an auction that interests him/her. The Auction system can have many Customers concurrently the auction using site at any one time.
Primary Actor: Customer
Precondition: The Customer has already identified him/herself to the System.
Main Success Scenario:
Steps 1-2 can be repeated according to the intent of the User.
1. Customer requests System for auctions on a particular item or item group.
2. System provides a view of the information pertaining to the request.
Extensions:
(1-2)a. (at any time) Customer requests System to go back to his previous selection.
(1-2)a.1. System provides a view of the information requested previously; use case continues at step 1.
1a. Customer refines his selection; use case continues at step 2.

A.2 Auction System: Task 2 (cf. Sect. 4.2)

The Auction System has any number of context entities all of the type Person (anyone accessing the system by Internet). A Person may play three roles: User (plain Internet user), Customer (a User identified by the Auction System and connected with it) and Seller (a Customer selling some goods using the Auction System). We give a first version of the Context View showing the Auction System and the Persons (the incomplete cooperation diagram just shows that the Persons interact only with the Auction System).



A.3 Auction System: Outcome of Task 3

In this subsection we present the final outcome of task 3, i.e., a preliminary version of the Data View, the Context View and the interface of the Auction System, and the many problems about the Auction System found while performing this task.

Use case buy and sell goods by auction

Q: *Can a registration information be wrong,? or can it be only complete or incomplete ? Similar questions are treated differently in other use cases: indeed, we have insufficient/ wrong for credit transfer details, and insufficient/not acceptable for auction registration information.* **A:** *We assume that a registration information may be acceptable or not acceptable, where the acceptaibility will be defined by an accompanying document; similarly for credit transfer details and auction registration information.*

Q: *How do a User get her/his identification?* **A:** *We assume that such information will be given to the User by the Auction System when accepting her/his enrollement.*

. . . . . so 2. of buy and sell goods by auction should be changed by adding "and send/give to her/him an identification to be used to connect to the system in the future". . . . .

Q: *How can a Customer be recognized by the Auction System during a connection?* **A:** *We introduce "session keys" as a general mechanism to allow the Auction System to recognize an identified Customer during the successive interactions.*

Q: *In the use case description the word "exit" is used twice (6. and (4-5)||a), does it have the same meaning in the two cases?* **A:** *We think that "exit" in 6. means that a User cancels her/his enrolment to the Auction System, whereas in (4-5)||a) it means that the User ends the session with the Auction System. To avoid confusion, we will use the word "leave" for the first case.*

Q: *Can the User "leave" the Auction System when (s)he decides to do that, and not only when the Auction System detects that it is inactive for some time?* **A:** *We think that it is safer to have also this possibility.*

Use case identify user **Q:** *What happen when a User gives an unacceptable identification is partly described in this use case and partly in buy and sell goods by auction, and consists of the Auction System informing the user that her/his identification is not valid and after prompting her/him to enrol.* **A:** *We decide that there is just a unique interaction of the Auction System with the User corresponding to signal the unacceptable identification and prompting to enrol; and we include this activity in this use case.*

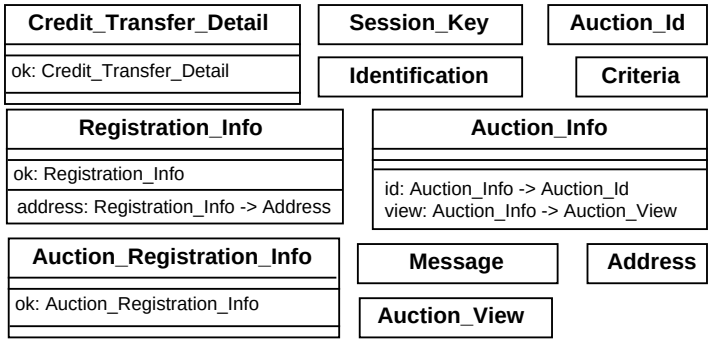
Use case increase credit **Q:** *There are many Customer's Credit Organizations, or just one?* **A:** *We assume that there is only one Customer's Credit Organization.*

To realize the use case increase credit the Auction System must interact with some "Customer's Credit Organization" not further detailed, this will be a new context entity to be added to the Context View.

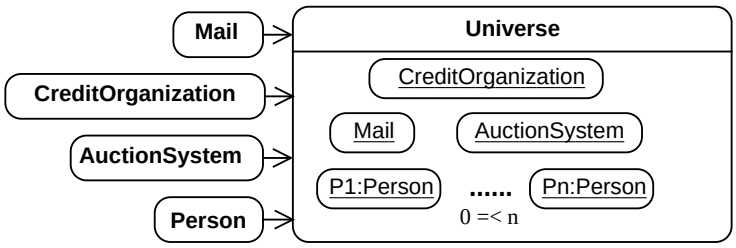
Use case search item **Q:** *How is the search of an item performed?* **A:** *We think that the Customer has to give the Auction System a search criteria, that is some information on how to perform the search for auctioned items; for example, by keywords or by categories, and in this last case which are the categories and how are they organized in a hierarchy.*

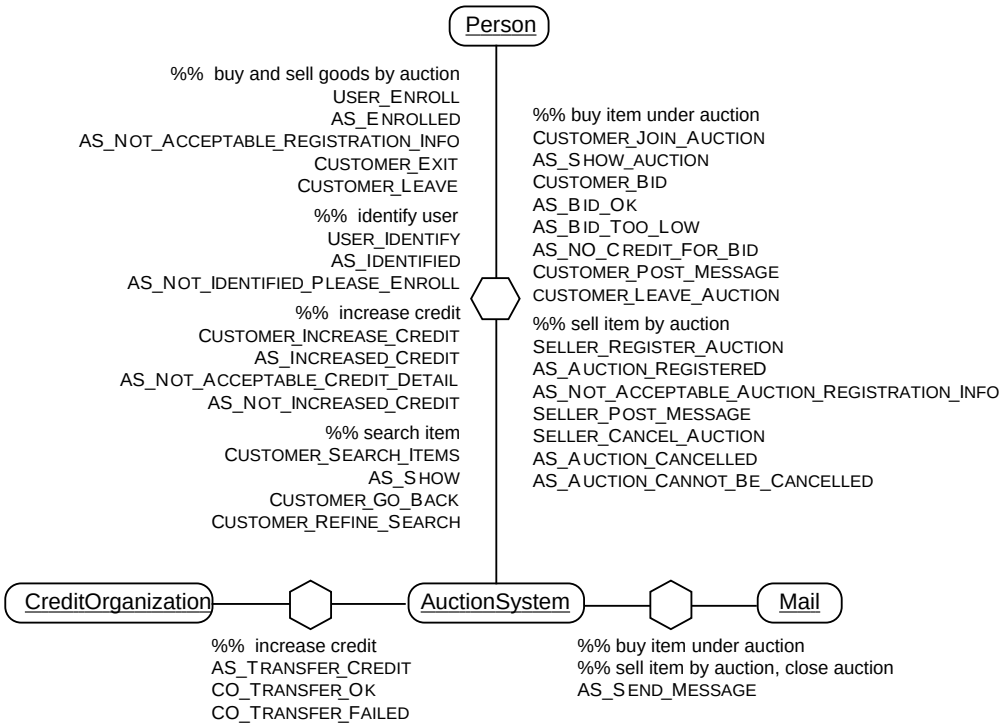
FG requirement specification at the end of task 3 The partial fragment of the various diagrams produced while examining the use cases are at the end put together. The result of this last activity is shown in the following pictures. The Data View is given first, for formatting reasons.

Data View



Context View





AuctionSystem Interface

AuctionSystem

%% buy and sell goods by auction
USER_ENROLL(Registration_Info)
AS_ENROLLED(Registration_Info,Identification)
AS_NON_ACCEPTABLE_REGISTRATION_INFO(Registration_Info)
CUSTOMER_EXIT(Session_Key)
CUSTOMER_LEAVE(Session_Key)

%% identify user
USER_IDENTIFY(Identification)
AS_IDENTIFIED(Identification,Session_Key)
AS_NOT_IDENTIFIED_PLEASE_ENROLL(Identification)

%% increase credit
CUSTOMER_INCREASE_CREDIT(Session_Key,Credit_Transfer_Detail)
AS_TRANSFER_CREDIT(Credit_Transfer_Detail)
CO_TRANSFER_OK(Credit_Transfer_Detail)
CO_TRANSFER_FAILED(Credit_Transfer_Detail)
AS_INCREASED_CREDIT(Session_Key)
AS_NON_ACCEPTABLE_CREDIT_DETAIL(Session_Key,Credit_Transfer_Detail)
AS_NOT_INCREASED_CREDIT(Session_Key)

%% search item
CUSTOMER_SEARCH_ITEMS(Session_Key,Criteria)
AS_SHOW(Session_Key,List(Item_View))
CUSTOMER_GO_BACK(Session_Key)
CUSTOMER_REFINE_SEARCH(Session_Key,Criteria)

%% buy item under auction
CUSTOMER_JOIN_AUCTION(Session_Key)
AS_SHOW_AUCTION(Session_Key,Auction_View)
CUSTOMER_BID(Session_Key,Auction_Id,Int)
AS_BID_OK(Session_Key,Auction_Id,Int)
AS_BID_TOO_LOW(Session_Key,Auction_Id,Int)
AS_NO_CREDIT_FOR_BID(Session_Key,Auction_Id,Int)
CUSTOMER_POST_MESSAGE(Session_Key,Auction_Id,Message)
CUSTOMER_LEAVE_AUCTION(Session_Key,Auction_Id)

%% sell item by auction
SELLER_REGISTER_AUCTION(Session_Key,Auction_Registration_Info)
AS_AUCTION_REGISTERED(Session_Key,Auction_View)
AS_NOT_ACCEPTABLE_AUCTION_REGISTRATION_INFO(Session_Key,Auction_Registration_Info)
SELLER_POST_MESSAGE(Session_Key,Auction_Id,Message)
SELLER_CANCEL_AUCTION(Session_Key,Auction_Id)
AS_AUCTION_CANCELLED(Session_Key,Auction_Id)
AS_AUCTION_CANNOT_BE_CANCELLED(Session_Key,Auction_Id)

%% buy item under auction, sell item by auction, close auction
AS_SEND_MESSAGE(Address,String)

%% buy and sell goods by auction, identify user
is_Registered(Registration_Info, Identification)

%% buy and sell goods by auction, identify user, increase credit , search item
%% buy item under auction, sell item by auction
is_Identified(Identification,Session_Key)

%% increase credit ,buy item under auction, sell item by auction, close auction
credit(Identification): Int

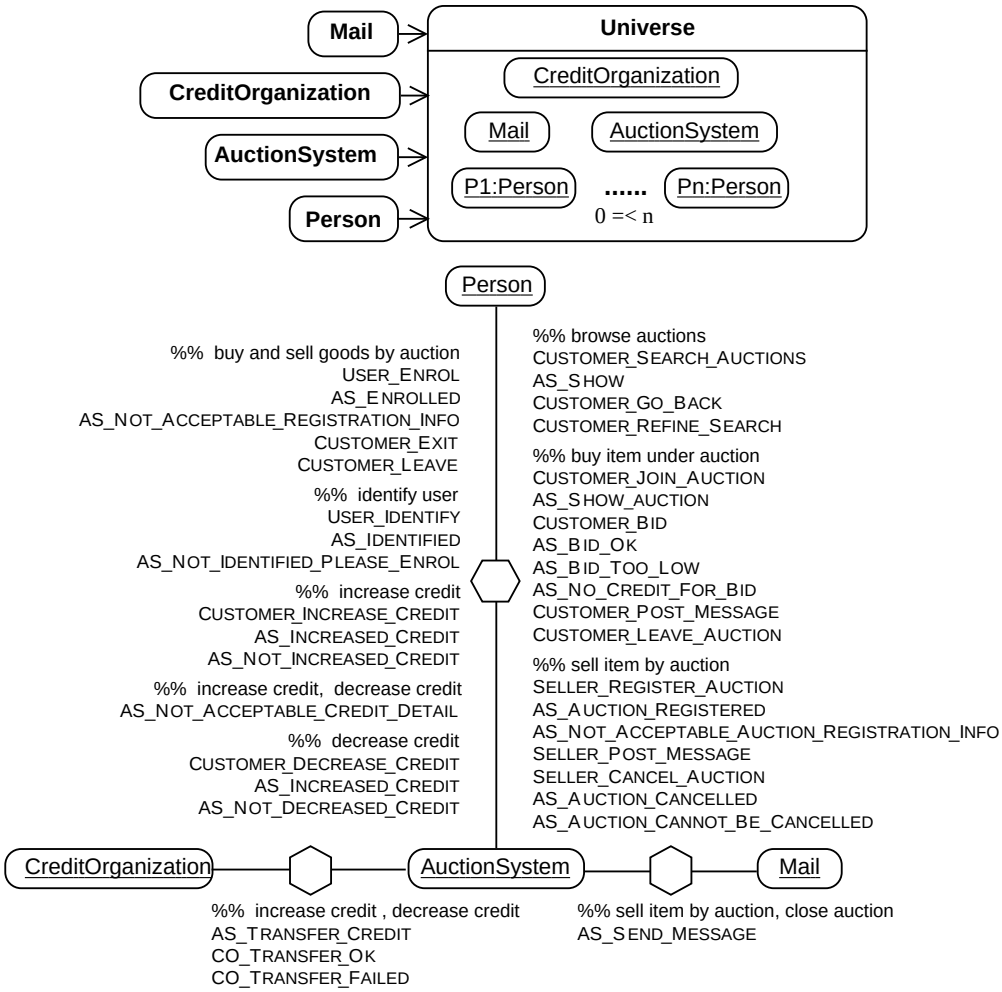
%% search item, buy item under auction, sell item by auction, close auction
infoAbout(Auction_Id): Auction_Info

A.4 Auction System: Task 4

This task consists in finding the properties about the Auction System by filling the tableaux generated by the elementary interactions and state observers found in the previous task, and by completing the specifications of the found data structures. Clearly, while doing this activity new state observers may be introduced, which will have then to be introduced in the tableaux and considered while looking for the properties; and the original use case based specification may be modified by reflecting the better insights on the Auction System gained while looking for properties. As a consequence, also the elementary interactions, the state observers and the data structures may be modified to specify the new use cases; and again the tableaux has to be updated.

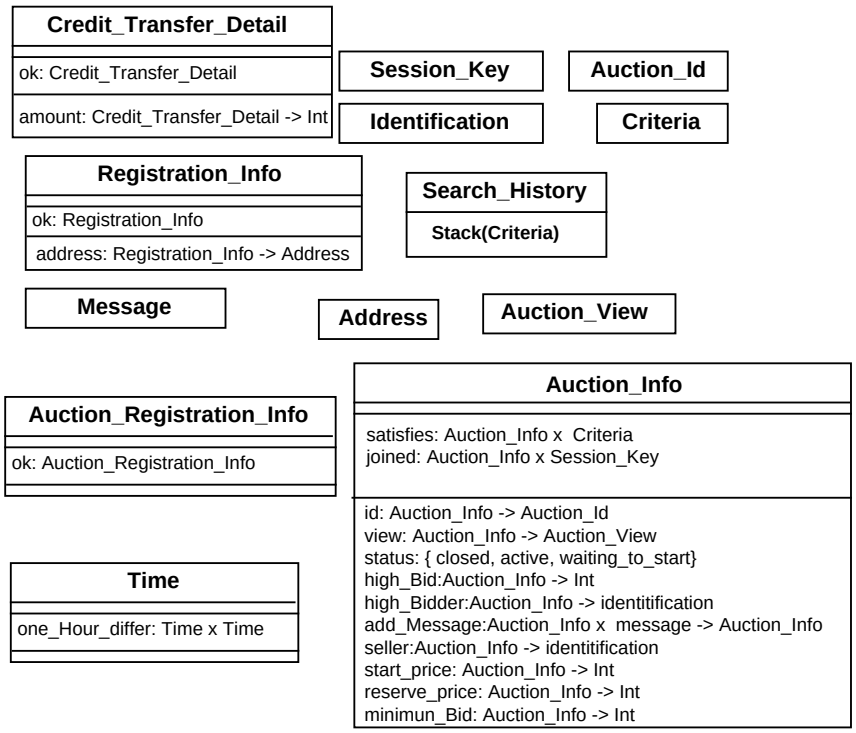
Here we report the final versions of the FG requirement specification the Auction System (the AuctionSystem interface, showing its elementary interactions and state observers, the Data View, the Context View, and the properties) corresponding to the new use case based specification reported in Sect. A.5.

Context View



AuctionSystem Interface

Data View



The Properties Here the properties are simply grouped following the rows of the tableaux. Together with the properties we report also the various questions arisen while looking for them, and how they have been settled.

To improve the readability of the properties we will use some derived state observers, i.e., new state observers which can be defined by combining the original ones.

- *unused*: *Identification*
  unused(*id*) **iff not exists** *ri*:*Registration_Info* **s.t.** *is_Registered*(*ri*,*id*)
- *unusedKey*: *Session_Key*
  unusedKey(*sk*) **iff not exists** *id*:*Identification* **s.t.** *is_Identified*(*id*,*sk*)
- *validKey*: *Session_Key*
  validKey(*sk*) **iff exists** *id*:*Identification* **s.t.** *is_Identified*(*id*,*sk*)
- *identityOf*(*Session_Key*): *Identification*
  identityOf(*sk*) = *id* **iff** *is_Identified*(*id*,*sk*)

Elementary Interaction USER_ENROLL
Q: *Does the acceptability of a registration information depend on the status of the Auction System?* **A:** *No. Notice that this means that the same User may enroll twice with the Auction System, or that two Users may use the same credit card or the same mail address to enroll.*

If a User asks the Auction System to be enrolled giving some registration information, then either
the information is acceptable and the Auction System informs the User that (s)he has been enrolled by sending her/him an unused identification to be used in the future connections with the Auction System
or
the information is not acceptable and the Auction System informs the user that (s)he has given an unacceptable information.
if USER_ENROLL(ri) **happen then**
($is\_Ok(ri)$ **and**
exists id: *Identification* **s.t.** $unused(id)$ **and in any case next** AS_ENROLLED(ri,id) **happen**)
or
(**not** $is\_Ok(ri)$ **and**
in any case next AS_NOT_ACCEPTABLE_REGISTRATION_INFO(ri) **happen**)

The Auction System cannot refuse for ever to accept enrollment requests from a User.
in one case eventually USER_ENROLL(ri)

Elementary Interaction AS_ENROLLED

The Auction System cannot say simultaneously to a User that has been enrolled and that the given registration information is unacceptable.
AS_ENROLLED(ri,id) **incompatible with** AS_NOT_ACCEPTABLE_REGISTRATION_INFO(ri)

An identification cannot be used to enroll simultaneously two different Users.
AS_ENROLLED(ri_1,id) **incompatible with** AS_ENROLLED(ri_2,id)

A User cannot be enrolled simultaneously with two different identifications.
AS_ENROLLED(ri,id_1) **incompatible with** AS_ENROLLED(ri,id_2)

Q: *Can a User be automatically enrolled in the Auction System?* **A:** *No, a registration happens only as a consequence of a request of the User.*

If the Auction System informs a User that (s)he has been enrolled, then
the User asked the Auction System to be enrolled,
the given registration information is correct, the sent identification is unused and
after the User will be registered with the sent identification.
if AS_ENROLLED(ri,id) **happen then**
in any case before USER_ENROLL(ri) **happened and**
$is\_Ok(ri)$ **and** $unused(id)$ **and** $is\_Registered^{nxt}(ri,id)$

Elementary Interaction AS_NOT_ACCEPTABLE_REGISTRATION_INFO

If the Auction System informs a User that the given registration information is not acceptable, then
the User asked the Auction System to be enrolled, and
the given registration information is not acceptable.
if AS_NOT_ACCEPTABLE_REGISTRATION_INFO(ri) **happen then**
in any case before USER_ENROLL(ri) **happened and not** $is\_Ok(ri)$

Elementary Interaction CUSTOMER_EXIT

If a User exits the Auction System (by her/his own decision), then
the User was identified and after (s)he is no more identified.
if CUSTOMER_EXIT(*sk*) **happen then**
exists *id*: *Identification* **s.t.**
is_Identified(*id*,*sk*) **and not** $is\_Identified^{nxt}(id,sk)$

It is possible that a User will never exit the Auction System.

Elementary Interaction CUSTOMER_LEAVE_SYSTEM
Q: *Can a User leave the Auction System at any time?* **A:** *A User cannot leave the Auction System when (s)he is the seller or has the highest bid in an active auction.*
Q: *What happens to an auction not yet started whose seller leaves the Auction System?*
A: *Such auction will be cancelled.*
Q: *What happens to the credit of a User who leaves the Auction System?* **A:** *That credit will be transferred to the Auction System owner*[5].

If a User leaves the Auction System, i.e., (s)he cancels her/his registration, then
the User was identified,
there are no active auctions where (s)he is either the seller or has the highest bid, and
after
(s)he is neither identified nor registered,
there are no auctions where (s)he is the seller waiting to start.
if CUSTOMER_LEAVE_SYSTEM(*sk*) **happen then**
exists *id*: *Identification* **s.t.**
is_Identified(*id*,*sk*) **and**
(**not exists** *aid*: *Auction_Id* **s.t.**
$status(infoAbout(aid))=active$ **and**
$(seller(infoAbout(aid)) = id$ **or** $high\_Bidder(infoAbout(aid)) = id))$ **and**
not $is\_Identified^{nxt}(id,sk)$ **and**
(**for all** *ri*: *Registration_Info* • **not** $is\_Registered^{nxt}(ri,id)$) **and**
not exists *aid*: *Auction_Id* **s.t.**
$(status(infoAbout(aid)) = not\_started$ **and** $seller(infoAbout(aid)) = id)$

It is possible that a User will never leave the Auction System.

Elementary Interaction USER_IDENTIFY
Q: *Will the Auction System allow for multiple simultaneous sessions of the same User?*
A: *Yes.*

If a User asks the Auction System to be identified, then
either
there exists a registration for the given identification,
the Auction System informs the User that (s)he has been identified and
send her/him an unused session key,
or
there does not exist a registration for the given identification and

[5] Here for simplicity do not explicitly model how the Auction System transfers money to its owner.

the Auction System informs the User that (s)he has not been identified and asks her/him to enroll.

if USER_IDENTIFY(*id*) **happen then**
(**exists** *ri*: *Registration_Info* **s.t.** *is_Registered*(*ri*,*id*) **and**
exists *sk*: *Session_Key* **s.t.**
unusedKey(*sk*) **and in any case next** AS_IDENTIFIED(*id*,*sk*) **happen**)
or
((**not exists** *ri*: *Registration_Info* **s.t.** *is_Registered*(*ri*,*id*)) **and**
in any case next AS_NOT_IDENTIFIED_PLEASE_ENROLL(*id*) **happen**)

The Auction System cannot refuse for ever to accept identification requests from the Users.
in one case eventually USER_IDENTIFY(*id*)

Elementary Interaction AS_IDENTIFIED

The Auction System cannot say simultaneously to a User that is identified and not identified.
AS_IDENTIFIED(*id*,*sk*) **incompatible with** AS_NOT_IDENTIFIED_PLEASE_ENROLL(*id*)

If the Auction System informs a User that (s)he has been identified sending her/him a session key, then
(s)he asked to be identified,
was registered,
the sent key is unused, and after (s)he will be identified.

if AS_IDENTIFIED(*id*,*sk*) **happen then**
in any case before USER_IDENTIFY(*id*) **happened and**
exists *ri*: *Registration_Info* **s.t.** *is_Registered*(*ri*,*id*) **and**
unusedKey(*sk*) **and** $is\_Identified^{nxt}(id,sk)$

Elementary Interaction AS_NOT_IDENTIFIED_PLEASE_ENROLL

If the Auction System informs a User that (s)he has not been identified, then
(s)he had asked to be identified and was not registered.

if AS_NOT_IDENTIFIED_PLEASE_ENROLL(*id*) **happen then**
in any case before USER_IDENTIFY(*id*) **happened and**
not exists *ri*: *Registration_Info* **s.t.** *is_Registered*(*ri*,*id*)

Elementary Interaction CUSTOMER_INCREASE_CREDIT

If the Customer requires to increase her/his credit by giving details for the credit transfer, then
s(he) is identified by the Auction System by a valid session key, and
either
the credit transfer details are acceptable,
the Auction System requires the credit transfer to the Credit Organization, and
either
the Credit Organization answers that the transfer is ok, and
the Auction System informs the Customer that her/his credit has been increased;
or
the Credit Organization answers that the transfer is failed, and
the Auction System informs the Customer that her/his credit has not been increased;
or
the credit transfer details are not acceptable and
the Auction System inform the Customer that the given credit transfer details are not acceptable.

if CUSTOMER_INCREASE_CREDIT(*sk*,*ctd*) **happen then**
  validKey(*sk*) **and**
    is_Ok(*ctd*) **and in any case**
    next AS_TRANSFER_CREDIT(*ctd*) **happen and**
      next CO_TRANSFER_OK(*ctd*) **happen and**
        next AS_INCREASED_CREDIT(*sk*) **happen**
       or
      next CO_TRANSFER_FAILED **and**
        next AS_NOT_INCREASED_CREDIT(*sk*) **happen**
  or
    not *is_Ok*(*ctd*) **and**
      in any case next AS_NOT_ACCEPTABLE_CREDIT_DETAIL(*sk*,*ctd*) **happen**

The Auction System cannot refuse for ever to accept requests to increase her/his credit from a User.
in one case eventually CUSTOMER_INCREASE_CREDIT(*sk*,*ctd*) **happen**

Elementary Interaction AS_TRANSFER_CREDITNote that the following properties make clear that the Auction System can get from/give to money a Costumer only in the case of an explicit request from her/him. Moreover, it is assumed that the Credit Organization always answers to a credit transfer request (it is never down).
Q: *When the Auction System performs a credit transfer with the Credit Organization, are the money effectively transferred to the Auction System owner or they are just blocked (that is, that money are still in the original bank account earning interest for the Customer or in the account of the Auction System owner earning interests for it?*
A: *We assume that they are effectively transferred tot he Auction System owner.*

If the Auction System asks a credit transfer to the Credit Organization, then
  a Customer identified by a valid key asked to have her/his credit increased or decreased; and
  after the Credit Organization will answer
    either by saying that the transfer is ok or that it has failed.
if AS_TRANSFER_CREDIT(*ctd*) **happen then**
  exists *sk* **s.t.** *validKey*(*sk*) **and**
    in any case before CUSTOMER_INCREASE_CREDIT(*sk*,*ctd*) **happened**
     or before CUSTOMER_DECREASE_CREDIT(*sk*,*ctd*) **happened and**
  in any case next CO_TRANSFER_OK(*ctd*) **happen or next** CO_TRANSFER_FAILED(*ctd*) **happen**

Elementary Interaction CO_TRANSFER_OK

The Credit Organization cannot simultaneously say that a credit transfer is ok and failed.
CO_TRANSFER_OK(*ctd*) **incompatible with** CO_TRANSFER_FAILED(*ctd*)

If the Credit Organization says that a credit transfer is ok, then
  the Auction System asked that credit transfer.
if CO_TRANSFER_OK(*ctd*) **happen then**
  in any case before AS_TRANSFER_CREDIT(*ctd*) **happened**

Elementary Interaction CO_TRANSFER_FAILED

If the Credit Organization says that a credit transfer is failed, then
  the Auction System asked that credit transfer.

if CO_Transfer_Failed(ctd) **happen then**
in any case before AS_Transfer_Credit(ctd) **happened**

Elementary Interaction AS_Increased_Credit

The Auction System cannot simultaneously say to a Customer that her/his credit has been increased and not increased.
AS_Increased_Credit(ctd) **incompatible with** AS_Not_Increased_Credit(ctd)

If the Auction System informs a Customer that her/his credit has been increased, then
in any case before
the Auction System received an ok message from the Credit Organization, and
the Customer asked to have her/his credit increased:
moreover, after
the credit of Customer has been increased by the amount defined in the credit transfer details.
if AS_Increased_Credit(sk, ctd) **happen then**
in any case before CO_Transfer_Ok(ctd) **happened and**
before Customer_Increase_Credit(sk, ctd) **happened**
and
$credit^{nxt}(identityOf(sk)) = credit(identityOf(sk)) + amount(ctd)$

Elementary Interaction AS_Not_Increased_Credit

If the Auction System informs a Customer that her/his credit has not been increased, then
before the Auction System received a failed transfer message from the Credit Organization, and
before the Customer asked to have her/his credit increased.
if AS_Not_Increased_Credit(sk, ctd) **happen then**
in any case
before CO_Transfer_Failed(ctd) **happened and**
before Customer_Increase_Credit(sk, ctd) **happened**

Elementary Interaction AS_Not_Acceptable_Credit_Detail. . . . . .

Elementary Interaction Customer_Decrease_Credit

If the Customer requires to decrease her/his credit by giving details for the credit transfer, then
s(he) is identified by the Auction System by a valid session key, and
either
the credit transfer details are acceptable and her/his credit is sufficient,
the Auction System requires the credit transfer to the Credit Organization, and
either
the Credit Organization answers that the transfer is ok, and
the Auction System informs the Customer that her/his credit has been decreased;
or
the Credit Organization answers that the transfer is failed, and
the Auction System informs the Customer that her/his credit has not been decreased;
or
the credit transfer details are not acceptable and
the Auction System inform the Customer that the given credit transfer details are not acceptable
or

her/his credit is not sufficient and
the Auction System inform the Customer of that.
if CUSTOMER_DECREASE_CREDIT(sk, ctd) **happen then**
$validKey(sk)$ **and**
$is\_Ok(ctd)$ **and** $credit(identityOf(sk)) \geq amount(ctd)$ **and in any case**
next AS_TRANSFER_CREDIT(ctd) **happen and**
next CO_TRANSFER_OK(ctd) **happen and**
next AS_INCREASED_CREDIT(sk) **happen**
or
next CO_TRANSFER_FAILED **and**
next AS_NOT_INCREASED_CREDIT(sk) **happen**
or
not $is\_Ok(ctd)$ **and**
in any case next AS_NOT_ACCEPTABLE_CREDIT_DETAIL(sk, ctd) **happen**
or
$credit(identityOf(sk)) < amount(ctd)$ **and**
in any case next AS_NOT_ENOUGH_CREDIT(sk) **happen**

The Auction System cannot refuse for ever to accept requests to decrease hers/his credit from a User.
in one case eventually CUSTOMER_DECREASE_CREDIT(sk, ctd) **happen**

Elementary Interaction AS_NOT_ENOUGH_CREDIT

If the Auction System informs a Customer that her/his credit is not enough for the required credit transfer, then
in any case
before the Customer asked to have her/his credit decreased and her/his credit was not enough.
if AS_NOT_ENOUGH_CREDIT(sk, ctd) **happen then**
in any case
before CUSTOMER_DECREASE_CREDIT(sk, ctd) **happened' and**
$credit(identityOf(sk)) < amount(ctd)$

Elementary Interaction AS_DECREASED_CREDIT

The Auction System cannot simultaneously say to a Customer that her/his credit has been decreased and not decreased.
AS_DECREASED_CREDIT(ctd) **incompatible with** AS_NOT_DECREASED_CREDIT(ctd)

If the Auction System informs a Customer that her/his credit has been decreased, then
in any case before
the Auction System received an ok message from the Credit Organization, and
the Customer asked to have her/his credit decreased:
moreover, after
the credit of Customer has been decreased by the amount defined in the credit transfer details.
if AS_DECREASED_CREDIT(sk, ctd) **happen then**
in any case before CO_TRANSFER_OK(ctd) **happened and**
before CUSTOMER_DECREASE_CREDIT(sk, ctd) **happened**
and
$credit^{nxt}(identityOf(sk)) = credit(identityOf(sk)) - amount(ctd)$

Elementary Interaction AS_Not_Decreased_Credit

If the Auction System informs a Customer that her/his credit has not been decreased, then
in any case
before the Auction System received a failed transfer message from the Credit Organization, and
before the Customer asked to have her/his credit decreased.
if AS_Not_Decreased_Credit(sk, ctd) **happen then**
in any case
before CO_Transfer_Failed(ctd) **happened and**
before Customer_Decrease_Credit(sk, ctd) **happened**

Elementary Interaction Customer_Search_Auctions
Q: *In which case is the search history reset?* **A:** *We assume that the search history is never reset for all the time a Customer is logged in the Auction System, and moreover we assume that it is unbound.*

If a Customer searches the auctions, then
(s)he is identified by a valid key; and
after the Auction System shows the auctions matching the given criteria,
those auctions become selected, and the search criteria is added to the history search.
if Customer_Search_Auctions($sk, crit$) **happen then**
$validKey(sk)$ **and**
exists $aids$: $set(Auction\_Id)$ **s.t.**
for all aid: $Auction\_Id \bullet (satisfies(infoAbout(aid), crit)$ **iff** $aid \in aids)$ **and**
in any case next AS_Show(sk, $\{view(infoAbout(aid)) \mid aid \in aids\}$) **happen and**
$selected\_Auctions^{nxt} = aids$ **and**
$search\_History^{nxt} = push(search\_History, crit)$

Elementary Interaction AS_Show

If the Auction System shows a set of auctions to a Customer, then
(s)he has searched them.
if AS_Show($sk, aviews$) **happen then**
exists $crit$:$Criteria$ **s.t.**
in any case before Customer_Search_Auctions($sk, crit$) **happened and**
exists $aids$: $set(Auction\_Id)$ **s.t.**
$aviews = \{view(infoAbout(aid)) \mid aid \in aids\}$ **and**
$(satisfies(infoAbout(aid), crit)$ **iff** $aid \in aids)$

Elementary Interaction Customer_Refine_Search
Q: *Is it the refinement of a search based on a kind of customer-related caching mechanism? i.e., the set of auction info currently found are searched using the new criteria?* **A:** *No, we assume that when a search is refined, a new search is made looking for the auctions satisfying both the old criteria and the new one.*

If a Customer asks the Auction System to refine the search, then
(s)he is identified by a valid key, a previous search has been performed, and after
all the auctions satisfying the both the new and the old criteria are shown to
the Customer.
if Customer_Refine_Search($sk, crit$) **happen then**
$validKey(sk)$ **and** $search\_History \neq empty$ **and**

exists *aids*: $set(Auction\_Id)$ **s.t.**
for all *aid*: $Auction\_Id$ •
$(satisfies(infoAbout(aid), conjunction(top(search\_History), crit))$ **iff** $aid \in aids)$ **and**
in any case next AS_SHOW(*sk*, $\{view(infoAbout(aid)) \mid aid \in aids\}$) **and**
$selected\_Auctions^{nxt} = aids$ **and**
$search\_History^{nxt} = push(search\_History, conjunction(top(search\_History), crit))$

Elementary Interaction CUSTOMER_GO_BACK
Q: *When a Customer requires the Auction System to go back, a new search is performed or the previous found data are shown again?* **A:** *We assume that a new search using the old criteria is performed.*

If a Customer asks the Auction System to go back to her/his previous search, then
(s)he is identified by a valid key a previous search has been performed, and
after
the Auction System shows the auctions matching the search criteria used previously,
and those auctions become selected and the history search is updated.
if CUSTOMER_GO_BACK(*sk*) **happen then**
$validKey(sk)$ **and** $search\_History \neq empty$ **and**
exists *aids*: $set(Auction\_Id)$ **s.t.**
for all *aid*: $Auction\_Id$ • $(satisfies(infoAbout(aid), top(search\_History))$ **iff** $aid \in aids)$ **and**
in any case next AS_SHOW(*sk*, $\{view(infoAbout(aid)) \mid aid \in aids\}$) **and**
$selected\_Auctions^{nxt} = aids$ **and**
$search\_History^{nxt} = pop(search\_History)$

Elementary Interaction CUSTOMER_JOIN_AUCTION
Looking for the precondition of CUSTOMER_JOIN_AUCTION we found the following unclear points about the Auction System.
Q: *Does the use case* search item *ends having selected one auction or one item? This is relevant because there may be many different auctions for the same item, e.g., a used car.* **A:** *The description of* search item *suggests some auctions, whereas that of* buy item under auction *suggests one item. We assume that* search item *ends with some selected auctions.*
Q: *Can the auction selected by the* search item *be in any status (e.g., closed or not yet started)?* **A:** *Yes, and this is quite sensible, since a Customer may be interested in knowing that some item has been sold in the past and at which price, or which are the current starting prices of some items, or that some items will be soon auctioned.*
Q: *Can a Customer try to join a closed or not-started auction?* **A:** *No, the Auction System should not provide this possibility, instead of answering with an error.*
The above problems lead us to *revise* the use case search item by discussing with the client. As a result, we now have the user goal level browse auctions use case ending with a selected group of auctions (indeed a Customer may be interested just in browsing all auctions handled by the Auction System). Moreover, the use case buy item under auction may start only when there is one selected auction, which is active. Then, we introduce a new state observer *selected_Auctions* that associates with each identified Customer (referred by a session key) the identities of the currently selected auctions.
Q: *Can a Customer join an auction to which (s)he is already joined?* **A:** *Yes, since there is no problem, also if a better choice may be that the Auction System send a warning to Customer.*

If a Customer joins an auction, then
(s)he is identified by a valid key,
Customer has selected exactly one auction, which is active;
and after
the Customer has joined that auction, and
the Auction System shows to her/him all the detail of the selected auction.
if CUSTOMER_JOIN_AUCTION(sk) **happen then**
$validKey(sk)$ **and**
exists aid:$Auction\_Id$ **s.t.**
$selected\_Auctions(sk) = \{aid\}$ **and** $status(infoAbout(aid)) = active$ **and**
$joined(sk, infoAbout(aid))$ **and**
in any case next AS_SHOW_AUCTION($sk, infoAbout(aid)$) **happen**

Elementary Interaction AS_SHOW_AUCTION

If the Auction System shows an auctions to a Customer, then
either (s)he has joined that auction,
or .....
if AS_SHOW_AUCTION(sk, ai) **happen then**
in any case before CUSTOMER_JOIN_AUCTION(sk) **happened and**
exists aid **s.t.** $infoAbout(aid) = ai$ **and** $selected\_Auctions(sk) = \{aid\}$
or .....

Elementary Interaction CUSTOMER_BID

If a Customer makes a bid in an auction, then
(s)he is identified by a valid key,
has joined that auction, which is active;
and after
the credit of the Customer is sufficient, the bid meets the minimum increment, and either
the bid becomes the highest one,
the amount is secured from the Customer credit,
the previous standing bid, if any, on the auction is released,
the Customer is informed that her/his bid was ok,
and the new situation of the auction is shown to all the Customers joined to the auction;
or
the credit of the Customer is not sufficient, and after the Customer is informed of that
or
the credit of the Customer is sufficient, the bid does not meet the minimum increment,
and after the Customer is informed of that.
if CUSTOMER_BID($sk aid, i$) **happen then**
$validKey(sk)$ **and**
$joined(sk, infoAbout(aid))$ **and** $status(infoAbout(aid)) = active$ **and**
$i \leq credit(identityOf(sk))$ **and** $bid\_Ok(infoAbout(aid), i)$ **and**
$high\_Bidder(infoAbout^{nxt}(aid)) = identityOf(sk)$ **and** $high\_Bid(infoAbout^{nxt}(aid)) = i$ **and**
$credit^{nxt}(identityOf(sk)) = credit(identityOf(sk)) - i$ **and**
(**if def**$(high\_Bid(infoAbout^{nxt}(aid)))$ **then**
$credit^{nxt}(high\_Bidder(infoAbout^{nxt}(aid))) =$
$credit(high\_Bidder(infoAbout^{nxt}(aid))) + high\_Bid(infoAbout^{nxt}(aid))$) **and**
in any case next AS_BID_OK(sk) **and**
(**for all** $sk_1 \bullet$ **if** $joined(sk_1, infoAbout(aid))$ **then next** AS_SHOW_AUCTION($infoAbout^{nxt}(aid)$))
or

$i >$ *credit*(*identityOf*(*sk*)) **and**
in any case next AS_No_Credit_For_Bid(*sk*,*aid*,*i*) **and**
or
$i \leq$ *credit*(*identityOf*(*sk*)) **and not** *bid_Ok*(*infoAbout*(*aid*),*i*) **and**
in any case next AS_Bid_Too_Low(*sk*,*aid*,*i*)

Elementary Interaction AS_Bid_Ok

While looking for the postcondition of AS_Bid_Ok, which may be also about the future behaviour of the Auction System after having performed the elementary interaction, we detected the following problem.
Q: *Is true that a Customer joined to an auction is informed twice of each new bid, once by receiving a view of the auction with the new bid and once by some kind of message? Moreover, if a Customer places a bid, and after leaves the auction, will be her/him ever informed of a new higher bid? More generally, which is the intended duration of an auction? a few hours when the participants bid many times, and continuously look at the current view of the auction? or several days, when the participants from time to time place their bids and look at the situation of the auction?* **A:** *An auction handled by the Auction System should last a few hours with all participants logged on; thus there is no need to inform the joined customers and the seller of the various bids, because they continuously examine the current view of the auction that the Auction System keeps updated.*

If the Auction System informs a Customer that her/his bid is ok, then
the Customer has placed such bid,
(s)he had sufficient credit, and the bid met the minimum increment; and after
the bid is recorded,
the amount is secured by the Customer credit,
the security on the previous high bid is released (if any), and
the updated auction view is sent to all the Customers joined to the auction.
if AS_Bid_Ok(*sk*,*aid*,*i*) **happen then**
in any case before Customer_Bid(*sk*,*aid*,*i*) **happened and**
$i \leq$ *credit*(*identityOf*(*sk*)) **and**
bid_Ok(*infoAbout*(*aid*),*i*) **and**
high_Bidder(*infoAbout*^{nxt}(*aid*)) = *identityOf*(*sk*) **and** *high_Bid*(*infoAbout*^{nxt}(*aid*)) = *i* **and**
credit^{nxt}(*identityOf*(*sk*)) = *credit*(*identityOf*(*sk*)) - *i* **and**
(**if is defined** (*high_Bidder*(*infoAbout*(*aid*))) **then**
credit^{nxt}(*high_Bidder*(*infoAbout*(*aid*))) =
credit(*high_Bidder*(*infoAbout*(*aid*))) + *high_Bid*(*infoAbout*(*aid*))) **and**
for all sk_1:*Session_Key* •
if *joined*(*aid*,sk_1) **then** AS_Show_Auction(sk_1,*infoAbout*^{nxt}(*aid*))

Elementary Interaction AS_Bid_Too_Low

The Auction System cannot say simultaneously that a bid was too low and that the credit was insufficient.
AS_Bid_Too_Low(*sk*,*aid*,*i*) **incompatible with** AS_No_Credit_For_Bid(*sk*,*aid*,*i*)

If the Auction System informs a Customer that her/his bid is too low, then
the Customer has placed such bid and the bid has not met the minimum increment.
if AS_Bid_Too_Low(*sk*,*aid*,*i*) **happen then**

in any case before CUSTOMER_BID(sk, aid, i) **happened and**
not $bid\_Ok(infoAbout(aid), i)$

Elementary interaction AS_NO_CREDIT_FOR_BID

If the Auction System informs a Customer that her/his credit is insufficient, then
the Customer has placed such bid and
her/his credit is not sufficient.
if AS_NO_CREDIT_FOR_BID(sk, aid, i) **happen then**
in any case before CUSTOMER_BID(sk, aid, i) **happened and**
$i > credit(identityOf(sk))$

Elementary interaction CUSTOMER_POST_MESSAGE

If a Customer posts a message to an auction, then
(s)he is identified by a valid key,
has joined that auction, which is active;
and after the new situation of the auction with the posted message is shown to all the Customers joined to the auction.
if CUSTOMER_POST_MESSAGE($sk aid, mes$) **happen then**
$validKey(sk)$ **and**
$joined(sk, infoAbout(aid))$ **and** $status(infoAbout(aid)) = active$ **and**
$infoAbout^{nxt}(aid)) = add\_Message(infoAbout(aid), mes, sk)$
in any case for all sk_1 •
if $joined(sk_1, infoAbout(aid))$ **then next** AS_SHOW_AUCTION($infoAbout^{nxt}(aid)$)

Elementary interaction CUSTOMER_LEAVE_AUCTION

If a Customer leaves an auction, then
s(he) was joined to this auction, and after is not joined with it.
if CUSTOMER_LEAVE_AUCTION(sk, aid) **happen then**
$joined(sk, infoAbout(aid))$ **and not** $joined(sk, infoAbout^{nxt}(aid))$

State Observer *is_Registered*
Q: *Should the Auction System allows a User to unregister?* **A:** *Yes, but only when (s)he is not the seller in an active auction or has not the highest bid in an active auction.*
Q: *Can the Auction System cancels a User registration?* **A:** *No.*
Q: *Should the Auction System allow a User to change her/his identification?* **A:** *No.*

A registration information uniquely identifies a User inside the Auction System.
$is\_Registered(ri, id_1)$ **and** $is\_Registered(ri, id_2)$ **and** $id_1 \neq id_2$ **is impossible**

No two Users may be registered with the same identification.
$is\_Registered(ri_1, id)$ **and** $is\_Registered(ri_2, id)$ **and** $ri_1 \neq ri_2$ **is impossible**

A User becomes registered only after that the Auction System has enrolled her/him.
if not $is\_Registered(ri, id)$ **and** $is\_Registered^{nxt}(ri, id)$ **then** AS_ENROLLED(ri, id) **happen**

If a Customer becomes unregistered, then (s)he asked the Auction System to leave it.

if *is_Registered*(*ri*,*id*) **and not** *is_Registered*^{nxt}(*ri*,*id*) **then**
  exists *sk*:*Session_Key* **s.t.**
    identityOf(*sk*)=*id* **and** CUSTOMER_LEAVE_SYSTEM(*sk*) **happen**

A registered User cannot change her/his identification.
is_Registered(*ri*,id_1) **and** *is_Registered*^{nxt}(*ri*,id_2) **and** $id_1 \neq id_2$ **is impossible**

The value of the credit is defined for each registered User.
if *is_Registered*(*ri*,*id*) **then def**(*credit*(*id*))

State Observer *is_Identified*
Q: *Is it possible that a Customer establishes simultaneously two different sessions with the Auction System?* **A:** *Yes*
Q: *Should the Auction System provide some mechanism of automatic disconnection, perhaps after some time of inactivity it is sensible to disconnect a Customer.* **A:** *The Auction System deletes any connection inactive during the last hour.*

A session key cannot be used to identify two different Customers.
is_Identified(id_1,*sk*) **and** *is_Identified*(id_2,*sk*) **and** $id_1 \neq id_2$ **is impossible**

An identified Customer must be registered in the Auction System.
if *is_Identified*(*id*,*sk*) **then exists** *ri*: *Registration_Info* **s.t.** *is_Registered*(*ri*,*id*)

A User becomes identified only after receiving the corresponding message from the Auction System.
if not *is_Identified*(*id*,*sk*) **and** *is_Identified*^{nxt}(*id*,*sk*) **then**
  AS_IDENTIFIED(*id*,*sk*) **happen**

A User becomes unidentified when
  either (s)he exits the Auction System or (s)he has been inactive for one hour.
if *is_Identified*(*id*,*sk*) **and not** *is_Identified*^{nxt}(*id*,*sk*) **then**
  CUSTOMER_EXIT(*sk*) **happen or** *one_Hour_Differ*(*time*,*last_action*(*sk*))

State Observer *credit*
Q: *Can a Customer using the Auction System only for selling items collect her/his money? Moreover, can a buying Customer recover her/his money when (s)he is no more interested in buying?* **A:** *Yes; thus we have to add a new use case* decrease credit *for allowing a Customer to recover her/his credit.*

Only a registered Customer may have a credit.
if def(*credit*(*id*)) **then exists** *ri*:*Registration_Info* **s.t.** *is_Registered*(*ri*,*id*)

ACustomer cannot be in debit.
if def(*credit*(*id*)) **then** *credit*(*id*) ≥ 0

If the credit of a Customer increases, then
  either the Customer asked the Auction System to increase it,
  or a bid of the Customer has been overcame,
  or an auction where the Customer is the seller ended successfully.
if *credit*^{nxt}(*id*) = *credit*(*id*) + *i* **and** *i*> 0 **then**

exists *sk*:*Session_Key*, *ctd*:*Credit_Transfer_Detail* **s.t.**
is_Identified(*id*,*sk*) **and** *i*= *amount*(*ctd*) **and**
in any case AS_Increased_Credit(*sk*,*ctd*) **happen**
or
exists *aid*:*Auction_Id* **s.t.** *high_Bidder*(*infoAbout*(*aid*)) = *id* **and**
high_Bidder(*infoAbout*^{nxt}(*aid*)) ≠ *id* **and** *high_Bid*(*infoAbout*(*aid*)) = *i*
or
exists *aid*:*Auction_Id* **s.t.**
AS_Send_Message(*addressOf*(*id*),"closed auction", *aid*,*i*) **happen**

If the credit of a Customer decreases, then
either the Customer asked the Auction System to decrease it,
or the Customer made a bid in an auction.
if *credit*^{nxt}(*id*) = *credit*(*id*) - *i* **and** *i*> 0 **then**
exists *sk*:*Session_Key*, *ctd*:*Credit_Transfer_Detail* **s.t.**
AS_Decreased_Credit(*sk*,*ctd*) **happened and**
i= *amount*(*ctd*) **and** *is_Identified*(*id*,*sk*)
or exists *sk*:*Session_Key*, *aid*:*Auction_Id* **s.t.**
AS_Bid_Ok(*sk*,*aid*,*i*) **happened and** *is_Identified*(*id*,*sk*)

State Observer *infoAbout*
Q: *Can a Seller bids in her/his own auctions?* **A:** *No; but note the Auction System does not guarantee that the same person cannot enroll twice using two different identifications.*
Q: *Will the information about a closed auction preserved for ever?* **A:** *Yes, we think that allowing Customer to browse these historic information may be interesting (e.g., you can see how the price of some item changed along the time), and so to attract new users.*

If an auction is active, then
the actual time is after its starting time and before its closing time.
if *status*(*infoAbout*(*aid*)) = *active* **then**
start_Time(*infoAbout*(*aid*)) ≤ *time* **and** *time*≤ *close_Time*(*infoAbout*(*aid*))

If an auction is closed, then the actual time is after its closing time.
if *status*(*infoAbout*(*aid*)) = *closed* **then** *close_Time*(*infoAbout*(*aid*)) < *time*

If an auction is waiting to start, then the actual time is before its starting time.
if *status*(*infoAbout*(*aid*)) = *not_started* **then** *start_Time*(*infoAbout*(*aid*)) > *time*

The Customer having the highest bid in an auction cannot be the seller.
high_Bidder(*infoAbout*(*aid*)) ≠ *seller*(*infoAbout*(*aid*))

The information about a closed auction will never change.
infoAbout(*aid*) ≠ *infoAbout*^{nxt}(*aid*) **and** *status*(*infoAbout*(*aid*)) = *closed* **is impossible**

The seller, the initial price, the reserve price, the minimum bid increment, the starting and closing time of an auction cannot change.
seller(*infoAbout*(*aid*)) ≠ *seller*(*infoAbout*^{nxt}(*aid*)) **is impossible**
initial_Price(*infoAbout*(*aid*)) ≠ *initial_Price*(*infoAbout*^{nxt}(*aid*)) **is impossible**
reserve_Price(*infoAbout*(*aid*)) ≠ *reserve_Price*(*infoAbout*^{nxt}(*aid*)) **is impossible**

$minimum\_Increment(infoAbout(aid)) \neq minimum\_Increment(infoAbout^{nxt}(aid))$ **is impossible**
$start\_Time(infoAbout(aid)) \neq start\_Time(infoAbout^{nxt}(aid))$ **is impossible**
$close\_Time(infoAbout(aid)) \neq close\_Time(infoAbout^{nxt}(aid))$ **is impossible**

The information about a closed auction are preserved for ever.
if $status(infoAbout(aid)) = closed$ **then** $infoAbout^{nxt}(aid) = infoAbout(aid)$

The highest bid and its owner change only when a new bid is accepted by the Auction System.
if $high\_Bid(infoAbout(aid)) \neq high\_Bid(infoAbout^{nxt}(aid))$ **or**
$high\_Bidder(infoAbout(aid)) \neq high\_Bidder(infoAbout^{nxt}(aid))$
then
exists sk: $Session\_Key$ **s.t.** AS_BID_OK($sk, aid, high\_Bid(infoAbout^{nxt}(aid))$) **happen and**
$identityOf(sk) = high\_Bidder(infoAbout^{nxt}(aid))$

The seller and the owner of the highest bidder of an auction are registered in the Auction System.
$is\_Registered(seller(infoAbout(aid)))$ **and** $is\_Registered(high\_Bidder(infoAbout(aid)))$

A Customer joined with an auction, is identified by the Auction System.
if $joined(infoAbout(aid), sk)$ **then** $is\_Identified(sk)$

A.5 Auction System Task 5: New Use Case Based Requirement Specification

Here we report the new use case based specification of the requirement on the Auction System. Recall that in the following use case descriptions "**" means that the details about some aspects of the Auction System (e.g., data format or rules to follow to perform some activity) are given in an accompanying document, not given in [13] and thus not considered here.

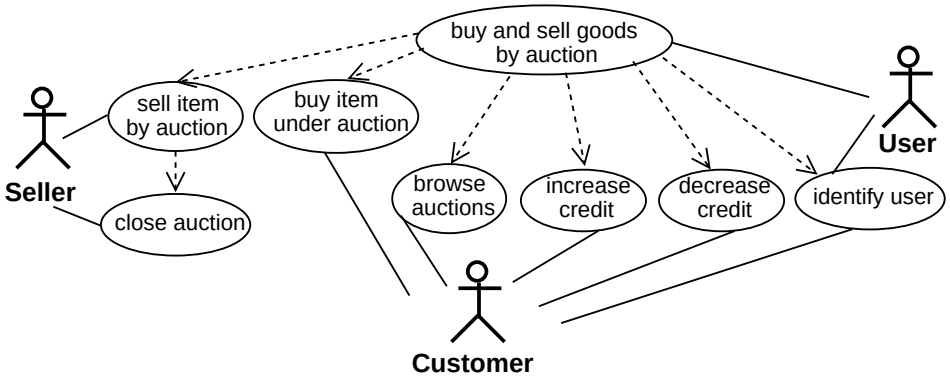


Fig. 3. Auction System: New use cases and actors

The use cases to be considered are: buy and sell goods by auction, identify user, increase credit, decrease credit (*new*), buy item under auction, sell item by auction, close auction, search item, and browse auctions (*new*).

Use Case buy and sell goods by auction

Level: Summary
Intention in Context: The intention of the User is to buy and sell goods by auctions over time. A User can be involved in multiple auctions at any one time.
Primary Actor: User (becomes Customer once (s)he has identified her—/himself with the System)
Main Success Scenario:
All Users must first enroll with the System before they have the right to use it
1. User enrolls with System, providing System with registration information**.
2. System validates the registration information and enrolls the User.
Steps 3-6 can be repeated according to the intention of the User to buy and sell over time
3. User identifies her/himself to System (identify user).
Steps 4-5 can be performed in parallel and individually repeated.
A customer may bid and sell in many auctions at any one time.
4. Customer increases/decreases her/his credit with System (increase credit and decrease credit).
5. Customer buys and sells items (buy item under auction, or sell item by auction).
6. Customer browses the auctions handled by System, closed, active or still to be started (browse auctions).
7. Customer exits System (i.e., (s)he ends the current session with System by her/his own decision); use case continues at step 3.
8. Whenever there are no active auctions where Customer is the seller or has the highest bid,

Customer leaves System, and
- System cancels the registration of Customer,
- any left credit of Customer is transferred to the Auction System owner, and
- any auction where Customer is the seller not yet started is cancelled.

Extensions:
2a. System ascertains that User did not provide acceptable information** to register her/him: (The acceptability of the information does not depend on the past history of the System, that is on which are the Users already registered; thus a person may register many times and will be considered as different Users by the System.)
  2a.1. System informs User; use case continues at step 1.
3a. System fails to identify User; use case continues at step 1.
(4-7)||a. System detects that Customer has left the System (defined by some sort of time-out on user response (open issue): use case continues at step 3.

Use Case identify user

Level: Subfunction
Intention in Context: The intention of the User is to identify her/himself so that (s)he may participate in auction(s).
Primary Actor: User
Main Success Scenario:
1. User identifies her/himself with System by providing identification**.
2. System validates the identification and sends back an unused identification.

Extensions:
2a. System fails to identify User:
  2a.1. System informs User and asks to enroll; use case ends in failure.

Use Case increase credit

Level: User Goal
Intention in Context: The intention of the User is to increase her/his credit with the System so that s(he) may buy goods on auction(s).
Primary Actor: Customer
Secondary Actor: The Credit Organization, which allows to transfer money from the Customer to the Auction System. It is assumed that the Credit Organization always answers to each credit transfer requests, either positively or negatively.
Precondition: The Customer has already identified her/himself to the System.
Main Success Scenario:
1. Customer requests System to increase her/his credit, and provides credit transfer details**.
2. System validates the transfer details.
3. System asks the Credit Organization for a credit transfer using the information provided by Customer.
4. System receives a positive answer from the Credit Organization, and informs Customer of a successful outcome.

Extensions:
2a. System ascertains that unacceptable information was provided:
  2a.1. System informs User; use case goes back to step 1.
4a. System receives a negative answer from the Credit Organization, and informs Customer of the failure; use case ends in failure.

Use Case decrease credit

Level: User Goal
Intention in Context: The intention of the User is to decrease her/his credit with the System so that s(he) may recovery the money earned by selling goods or advanced to the System for future buyings.
Primary Actor: Customer
Secondary Actor: The Credit Organization, which allows to transfer money from the Auction System to the Customer. It is assumed that the Credit Organization always answers to each credit transfer requests, either positively or negatively.
Precondition: The Customer has already identified her/himself to the System.
Main Success Scenario:
1. Customer requests System to decrease her/his credit, and provides credit transfer details**.
2. System validates the transfer details.
3. System asks the Credit Organization for a credit transfer using the information provided by Customer.
4. System receives a positive answer from the Credit Organization, and informs Customer of a successful outcome.

Extensions:
2a. System ascertains that non-acceptable information was provided:
  2a.1. System informs User; use case goes back to step 1.
2b. System ascertains that current credit of Customer is too low for the required transfer:
  2b.1. System informs User; use case goes back to step 1.
4a. System receives a negative answer from the Credit Organization, and informs Customer of the failure; use case ends in failure.

Use Case buy item under auction

Level: User Goal
Intention in Context: The intention of the Customer is to follow the auction, which may then evolve into an intention to buy an item by auction, i.e., (s)he may then choose to bid for an item. The Customer may bid in many different auctions at any one time.
Primary Actor: Customer
Precondition: The Customer has already identified her/himself to the System and selected one active auction.
Main Success Scenario:
Customer may leave the auction and come back again later to look at the progress of the auction, without effect on the auction; in this case, the Customer is required to join the auction again.
1. Customer requests System to join the selected auction.
2. System presents a view of the auction** to Customer.

Steps 3-4 can be repeated according to the intentions and bidding policy of the Customer
3. Customer makes a bid on the item to System.
4. System validates the bid, records it, removes the bid amount from Customer's credit**, returns the previous bid to the high bidder's credit (only when there was a previous standing bid), and updates the view of the auction for the item** with new high bid to all Customers that are joined to the auction.

Customer has the high bid for the auction
5. System closes the auction with a winning bid by Customer.

Extensions:
2a. The Customer is the Seller of the auction; System informs Customer that (s)he cannot join the auction. Use case ends with failure.
4a. Customer leaves auction:

4a.1a. System ascertains that Customer has highest bid in auction:
4a.1a.1. System continues auction without effect; use case continues at step 5
4a.1b. System ascertains that Customer does not have high bid in auction: use case ends in failure.
4||a. Customer requests System to post a message to auction and provides the message content**.
4||a.1. System updates the view of the auction with the added message to all Customers that are joined to the auction; use cases continues from where it was interrupted.
3a. System determines that bid does not meet the minimum increment**:
3a.1. System informs Customer; use cases continues at step 4.
3b. System determines that Customer does not have sufficient credit for the bid:
3b.1. System informs Customer; use cases continues at step 4.
5a. Customer was not the highest bidder:
5a.1. System closes the auction; use case ends in failure.

Use Case sell item by auction

Level: User Goal
Intention in Context: The intention of the Seller is to sell an item by auction. A Seller may be the seller in many auctions at any one time.
Primary Actor: Seller
Precondition: The Seller has already identified her/himself to the System.
Main Success Scenario:
Seller may leave the auction and come back again later to look at its progress, without effect on the auction.
1. Seller registers an item to sell on auction with System, providing registration information**.
2. System validates the information, presents a view of auction** to Seller, and records and publishes the new auction.
3. System closes auction (close auction) with a successful sale.
Extensions:
2a. System ascertains that it was not given acceptable information to define an auction:
2a.1. System informs Seller that certain details are not acceptable; use case continues at step 1.
(2-3)||b. Seller requests System to post a message to auction and provides the message content**.
(2-3)||b.1. System updates the view of the auction with the added message to all Customers that are joined to the auction; use cases continues from where it was interrupted.
3a. System closes auction without a successful sale: use case ends in failure.
3b. Seller requests System to cancel auction:
3b.1a. System detects that auction has already started:
3b.1a.1. System refuses**; use case continues from where it was interrupted.
3b.1b. System detects that auction has not yet started:
3b.1b.1. System cancels and removes auction, and publishes the fact that the auction is canceled; use case ends in failure.

Use Case close auction

Level: Subfunction
Intention in Context: The Seller proposed an expiration date for the auction that the System (on behalf of the Auction System Owners) agreed to**. The System closes down the auction at this moment.
Primary Actor: (Seller)
Trigger: The System detects that the auction has expired.
Main Success Scenario:
1. System detects that auction has expired.
2. System validates that the highest bid has met Seller's reserve price.
3. The System closes down auction:
   augments the Seller's credit by the winning bid amount minus the commission** taken for the auction service,
   informs the Customer with the winning bid that her/his bid was successful (open issue: this will most likely be done via email),
   informs all participants in the auction of the result, and
   informs the Seller of the result** and requests her/his to organize the delivery of the item with the Customer.

Extensions:
2a. System detects that highest bid has not met Seller's reserve price for the item:
   2a.1. System returns the highest bid to the the credit of the Customer with the highest bid, and it informs the participants and the Seller that there was no sale; use case ends in failure.
2b. System detects that no bid was made on auction:
   2b.1. System informs the participants and the Seller that there was no sale; use case ends in failure.

Use Case browse auctions

Level: User Goal
Intention in Context: The intention of the Customer is to browse the auctions that interests her/his, to find an item to buy, to see which are the current prices, to see what has been sold in the past, and to see which are the forthcoming auctions.
The Auction system can have many Customers concurrently using the auction site at any one time.
Primary Actor: Customer
Precondition: The Customer has already identified her/himself to the System.
Main Success Scenario:
Steps 1-2 can be repeated according to the intent of the User.
1. Customer requests System for auctions satisfying a given criteria**.
2. System provides an external view** of the auctions matching the criteria.

Extensions:
(1-2)a. (at any time) Customer requests System to go back to her/his previous selection.
   (1-2)a.1. System provides an up to date view of the information requested previously; use case continues at step 1.
1a. Customer refines her/his selection; use case continues at step 2.