INFORMATICA, 2001, Vol. 12, No. 3, 347–372

Verification Rules of Computerized Information System Model with Respect to Data Resource Processing

Rita BUTKIENĖ, Rimantas BUTLERIS
Kaunas University of Technology, Department of Information Systems
Studentų 50, 3027 Kaunas, Lithuania
e-mail: ribu@soften.ktu.lt, rimbut@if.ktu.lt

Received: April 2001

Abstract. This paper presents the set of verification rules to detect faults in specification of model of computerized information system. Computerized information system model is specified using Enterprise modeling approach, which integrates static and dynamic dependencies. Typical communication action loop is a basic construct of Enterprise modeling approach. Actions, which process data resources of computerized information system, are analyzed.

Key words: information system, requirement specification, enterprise modeling, integrity, static dependencies, and dynamic dependencies.

1. Introduction

The specification of requirements is one of the most important stages in the development of information system. The correct description is one of the principle goals in preparation of specification of requirements. A variety of methods for object-oriented systems analysis and design have been developed in the past years (Booch, 1994; Coad and Yourdan, 1991; Jacobson *et al.*, 1992; Martin and Odell, 1995; Rumbaugh *et al.*, 1991; Muller, 1997). Many methods rely on graphical specification technique. The reason is that concepts used in the method are easy to understand and use if they are expressed in graphical notation. Traditional methods use various graphical diagrams to define semantics of information systems. But each diagram is designed to describe one or few, but not all aspects of information system. It means that information system models should comprise a combination of several notations, each for some particular aspect. This leads to a difficult problem of integration of various graphical representations used to specify semantic of information systems. The Enterprise modeling approach suggested by R.Gustas (Gustas, 1997) was selected as having integration of static and dynamic dependencies. This approach provides also a uniform formal basis to analyze incompleteness, inconsistency and integrity of information system specification. However, Enterprise modeling approach as many others lack a complete set of verification rules to detect faults of specification. Detection of faults is very important because it enhances the quality of specification (Quality

of specification can be analyzed in terms of such characteristic features as consistency, cohesion, un-redundancy, ambiguity and completeness (Loucopoulos, 1992)). Well-done specification of information system is a significant support to create software with high functionality, low maintenance and adaptation costs. In this paper a set of verification rules is presented to detect faults of specification.

Many powerful tools already exist to assist in system development, for instance Oracle Designer/2000 (Barker, 1990; Barker and Longman, 1992), Rational Rose (Muller, 1997). But these tools sometimes are not useful, because they use technology of the user requirements specification, which is not adequate to the natural way of analysis of the user needs. Information systems designers, experts of information systems development use their knowledge and experience in their work (Cauvet *et al.*, 1991; Rolland and Cauvet, 1992). They perform the analysis of the user requirements and make formal specification. This paper is a part of research, in which the way the experts of information systems design follow in their work is analyzed and specified. This way of working was checked in practice and is used successfully.

Verification rules presented in this paper can be implemented in CASE-tool to facilitate the specification of information systems.

2. The Basic Constructs and Model of Specification

The way the experts of information system analysis and design follow in their work (Butkienė and Butleris, 1997) can be described as follows.

The specification of information system starts from the description of the purpose of computerized information system (CIS). CIS is specific subsystem of organizational system. The activity of the organization is based on the goals, which it wants to achieve. The activity of CIS is based on the goals, which users of CIS want to achieve with help of it. The basic purpose of CIS is – to process and convey information. What kind of information CIS must present to the user depends on the user needs. In other words, CIS is an assistant in the process of user goals achieving. The purpose of CIS can be very common, for example accounting, so it is important to detail this purpose up to certain reports, data flows. All of them will be named output reports. Output report is a result of functionality of CIS. Output reports define the applicability of CIS.

The next step after specification of output reports is a specification of data resources. Data resource is a resource of data (for example primary documents) to form the output reports. The user of CIS gives to system analyst-designer information concerning data resources: structure, presentation form, processing actions. The process of specification of output reports and preliminary structure of data resources and requirements adjusted to this process are discussed in (Butkienė and Butleris, 1998).

Specification of data resources is a background for modeling of entity-relationship model of problem domain. Specification of actions processing the data resources specifies not only the functionality of information system but gives an opportunity to specify data resources more precisely. Therefore, integration of static and dynamic dependencies

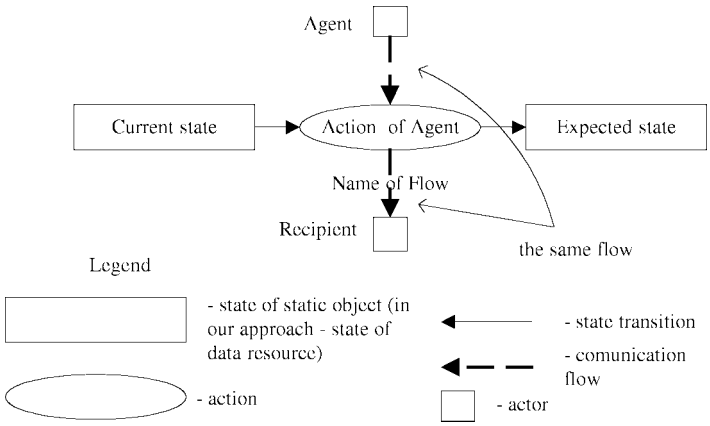

Fig. 1. Communication action and state transition dependency.

becomes necessary to get specification of higher quality. Enterprise modeling approach was selected for specification of user requirements for CIS because of it possibility to integrate static and dynamic dependencies. Typical communication action loop is a basic construct of enterprise modeling approach (Gustas, 1997). Communication action loop consists of communication actions and state transition dependencies. Graphical notation of communication action and state transition dependency is depicted in Fig. 1.

When only computerized information system is specified using Enterprise modeling approach (other subsystems of information system are not taken into account) then one of the actors of communication action is CIS and other – user of CIS (or other CIS if two different CIS are integrated). Graphical notation of typical communication action loop used in this paper is depicted in Fig. 2. This typical communication action loop specifies dependency of communication action between user and CIS, and dependency of state transition of data resource. Actions, which process data resources, are analyzed only. Actions, which process output reports, are not taken into account. Analysis of processes related with output reports of CIS is a subject of further research.

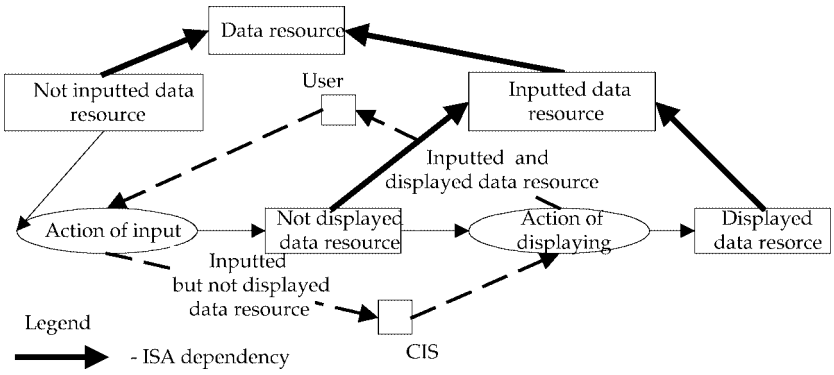

Fig. 2. Typical communication action loop.

This typical communication action loop represents a unit of work within a process of processing of data resources. Typical communication action loop includes:

- two actors – user and CIS;
- action performed by user – action of input (user is an agent, and CIS is a recipient);
- action performed by CIS – action of displaying (CIS is an agent, and user is a recipient);
- flow from user to CIS – inputted but not displayed data resource;
- flow from CIS to user – inputted and displayed data resource (for example, data displayed in the screen of computer, warning what data are saved);
- current state of data resource with respect to action performed by user – not inputted data resource;
- next state of data resource with respect to the action performed by user – inputted but not displayed data resource;
- current state of data resource with respect to action performed by CIS – inputted but not displayed data resource;
- next state of data resource with respect to the action performed by CIS – inputted and displayed data resource.

Typical communication action workflow loop shows that user is authorized to change the state of data resource from state "Not inputted data resource" to state "Inputted data resource" by using a predefined action of input. When user inputs data resource, then CIS is obliged to display inputted data resource. Modeling of states of data resources is related with modeling of structure of data resources. Inputted data resource has two states: "Not displayed" and "Displayed". These states and actions performed in respect to them are typical for each accepted by CIS data resource in a certain state. More attention is not given to them in this article. The actions of user require more attention because they are not trivial and vary in different problem domain. Further in this article the actions of user are analyzed only.

A formal definition of model M of specification of user requirements to CIS preliminary is defined as tuple $M =< O, I, T, P, A, S, H, R, Z >$, where

O – set of output reports,
I – set of data resources,
T – set of attributes,
P – set of actions,
A – set of actors,
S – set of states of data resource,
H – set of saved corrections of data resource in certain state,
R – set of relationships, $R = R_D \cup R_S$, where

R_D – set of relationships among data resources,

R_S – set of relationships among data resources being in a certain states,

Z – set of IS_A dependencies among possible states of data resources.

The first order predicate logic is used to specify dependencies between objects of sets specified above. The Table 1 is presented to describe the meanings of certain predicates.

The specified output reports define what kind of data resources must be specified. For each output report $\boldsymbol{o}$ at least one data resource $\boldsymbol{i}$ must be specified which is used to form the output report $\boldsymbol{o}$:

$$\forall o[O(o) \Rightarrow \exists i[I(i) \wedge use\_data\_resources(o,i)]].$$

Here: "$\Rightarrow$" means logical implication, "$\wedge$" means logical AND.

The attributes describe the structure of output reports, data resources, and states of data resources. For each output report $\boldsymbol{o}$ at least one attribute $\boldsymbol{t}$ must be specified which describes the structure of output report $\boldsymbol{o}$:

$$\forall o[O(o) \Rightarrow \exists t[T(t) \wedge describe\_o(o,t)]].$$

For each data resource $\boldsymbol{i}$ at least one attribute $\boldsymbol{t}$ must be specified which describes the data resource $\boldsymbol{i}$:

$$\forall i[I(i) \Rightarrow \exists t[T(t) \wedge describe\_i(i,t)]].$$

Each data resource $\boldsymbol{i}$ exist in a certain state $\boldsymbol{s}$. For each data resource $\boldsymbol{i}$ at least one state $\boldsymbol{s}$ must be specified:

$$\forall i[I(i) \Rightarrow \exists s[S(s) \wedge state\_of\_data\_resources(i,s)]].$$

Table 1

Description of predicates.

Description	Dependency to set	Predicate
o is an output report	$o \in O$	$O(o)$
i is a data resource	$i \in I$	$I(i)$
t is an attribute	$t \in T$	$T(t)$
p is an action	$p \in P$	$P(p)$
a is an actor	$a \in A$	$A(a)$
s is a state of data resource	$s \in S$	$S(s)$
h is saved correction of data resource	$h \in H$	$H(h)$
r is an association	$r \in R$	$R(r)$
r is an association between data resources	$r \in R_D$	$R_D(r)$
r is an association between data resources being in certain states	$r \in R_S$	$R_S(r)$
z is an IS_A dependency	$z \in Z$	$Z(z)$

Each state s is specified for only one data resource i:

$$\forall s[S(s) \Rightarrow \exists![I(i) \wedge state\_of\_data\_resources(i, s)]].$$

At least one attribute t describes each data resource i being in certain state s:

$$\forall i \forall s[[I(i) \wedge S(s) \wedge state\_of\_data\_resources(i, s)] \Rightarrow \exists t[T(t) \wedge describe\_state(i, s, t)]].$$

Each attribute t, which describe data resource i, describes state s of data resource i:

$$\forall i \forall t[[I(i) \wedge T(t) \wedge describe(i, t)] \Rightarrow \exists s[S(s) \wedge state\_of\_data\_resources(i, s) \wedge describe\_s(i, s, t)]].$$

States of certain data resource are related by IS_A relationship. Each IS_A relationship z relates two states s' and s'' (s' is a specialization of s'') specified for data resource i:

$$\forall z[Z(z) \Rightarrow \exists!\ i\ \exists! s' \exists! s''[I(i) \wedge S(s') \wedge S(s'') \wedge state\_of\_data\_resource(i, s') \wedge state\_of\_data\_resource(i, s'') \wedge is\_a(r, i, s', s'')]].$$

Each association r relates two data resources i' and i'', and it is not important in which state data resources are, or two data resources i' and i'' being in a certain states s' and s'' accordingly:

$$\forall r[R_D(r) \Rightarrow \exists! i' \exists! i''[I(i') \wedge I(i'') \wedge relate(r, i', i'')]],$$
$$\forall r[R_S(r) \Rightarrow \exists! i' \exists! i'' \exists! s' \exists! s''[I(i') \wedge I(i'') \wedge S(s') \wedge S(s'') \wedge relate(r, i', s', i'', s'')]].$$

For each action p an agent a' and recipient a'' must be specified:

$$\forall p[P(p) \Rightarrow \exists a' \exists a''[A(a') \wedge A(a'') \wedge agent(p, a') \wedge recipient(p, a'')]].$$

For each actor a at least one process p must be specified, in which an actor participate:

$$\forall a[A(a) \Rightarrow \exists p[P(p) \wedge [agent(p, a) \vee recipient(p, a)]]].$$

The data resource in each state must be associated with at least one action, which transfer data resource to this state. And each action must process at least one data resource being in a certain state. Associations among data resources and actions are discussed below in more detail.

3. Types of Actions

The actions of user can be of five types:

- action of creation – creates the item of data resource in a certain state;
- action of transference – transfers data resource from state to state;
- action of termination – deletes item of data resource being in a certain state;
- action of modification – modifies data resource without state changing;
- action of search – searches a specified data.

First four types of actions define life cycle of each data resource. The search of data resource is analogous to formation of output report, and is not discussed in this paper.

A lot of database management systems (DBMS) automated programming packages includes these actions in the menu of created CIS. But it is necessary to remember that these actions in the each CIS can be specific, performed with different limitations, which are determined by user requirements.

For each action $\boldsymbol{p}$ at least one data resource $\boldsymbol{i}$ being in state $\boldsymbol{s'}$, and which this action process, must be specified, and type of action must be defined. If action $\boldsymbol{p}$ is an action of transference, then one more state $\boldsymbol{s''}$ of data resource $\boldsymbol{i}$ must be specified. An action $\boldsymbol{p}$ can be one of type only respect to data resource $\boldsymbol{i}$. Constraints mentioned above are defined formally:

$$\forall p[P(p) \Rightarrow \exists i \exists s'[I(i) \wedge S(s') \wedge [[create(p,i,s') \wedge \neg\exists s''[S(s'') \wedge trasite(p,i,s',s'')] \wedge \neg delete(p,i,s') \wedge \neg modify(p,i,s')] \vee [\neg create(p,i,s') \wedge \exists s''[S(s'') \wedge trasite(p,i,s',s'')] \wedge \neg delete(p,i,s') \wedge \neg modify(p,i,s')] \vee [\neg create(p,i,s') \wedge \neg\exists s''[S(s'') \wedge trasite(p,i,s',s'')] \wedge delete(p,i,s') \wedge \neg modify(p,i,s')] \vee [\neg create(p,i,s') \wedge \neg\exists s''[S(s'') \wedge trasite(p,i,s',s'')] \wedge \neg delete(p,i,s') \wedge modify(p,i,s')]].$$

Action of Creation

This action starts the life cycle of data resource in CIS. The graphical notation of action of creation is depicted in Fig. 3. For the each data resource $\boldsymbol{i}$, at least one action $\boldsymbol{p}$ must be specified which creates data resource $\boldsymbol{i}$ in a certain state $\boldsymbol{s}$:

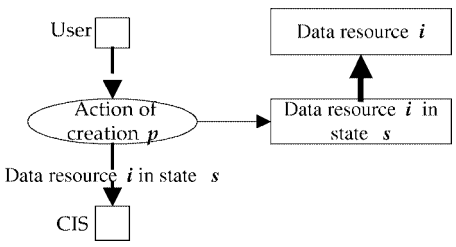

Fig. 3. The action of creation.

$$\forall i[I(i) \Rightarrow \exists p \exists s[P(p) \wedge S(s) \wedge create(p, i, s)]].$$

State $\boldsymbol{s}$ is the expected (next) state of data resource $\boldsymbol{i}$ according to action of creation $\boldsymbol{p}$:

$$\forall i \forall p \forall s[[I(i) \wedge P(p) \wedge S(s) \wedge create(p, i, s)] \Rightarrow next\_state(i, s, p)].$$

The current state $\boldsymbol{s'}$ of data resource $\boldsymbol{i}$ corresponding to action of creation $\boldsymbol{p}$ is not specified because it does not exist:

$$\forall i \forall p \forall s''[[I(i) \wedge P(p) \wedge S(s'') \wedge create(p, i, s'')] \Rightarrow \neg \exists s'[S(s') \wedge current\_state(i, s', p)]].$$

Different actions of creation, specified for the same data resource, create this data resource in the different states. This requirement helps to avoid redundancy in specification of processes.

Action of Transference

Action of transference $\boldsymbol{p}$ changes the state of certain data resource $\boldsymbol{i}$ from current state $\boldsymbol{s'}$ to a next state $\boldsymbol{s''}$:

$$\forall i \forall p \forall s' \forall s''[[I(i) \wedge P(p) \wedge S(s) \wedge transfer(p, i, s', s'')] \Rightarrow [current\_state(i, s', p) \wedge next\_state(i, s'', p)]].$$

The graphical notation of action of transference is depicted in Fig. 4. For each data resource several or none actions of transference can be specified.

Action of Termination

This action ends the life cycle of the data resource in the CIS. The graphical notation of action of termination is depicted in Fig. 5. The action of termination $\boldsymbol{p}$ deletes the item of data resource $\boldsymbol{i}$ from CIS. For each data resource $\boldsymbol{i}$ the action of termination $\boldsymbol{p}$ must be specified which deletes data resource $\boldsymbol{i}$ being in a certain state $\boldsymbol{s}$:

$$\forall i[I(i) \Rightarrow \exists p \exists s[P(p) \wedge S(s) \wedge delete(p, i, s)]].$$

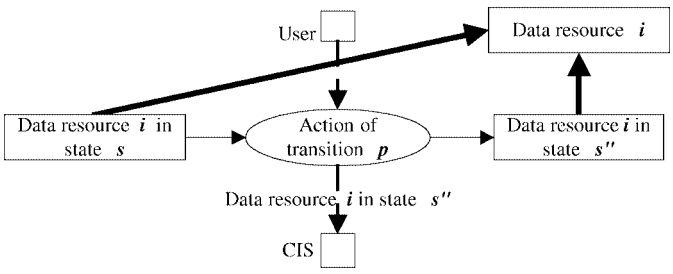

Fig. 4. The action of transference.

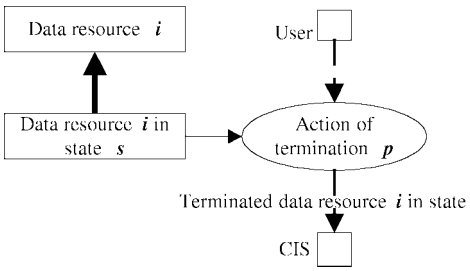

Fig. 5. The action of termination.

For each data resource several actions of termination can be specified. The user decides to which states (for example, s) of data resource i the action of termination p can be applied. The different actions of termination specified for the same data resource terminate the existence of the data resources being in the different states.

Only the current state s of data resource i must be specified for the action of termination p:

$$\forall i \forall p \forall s[I(i) \wedge P(p) \wedge S(s) \wedge delete(p, i, s)] \Rightarrow current\_state(i, s, p)].$$

The specification of next state is meaningless because it does not exist. For each data resource i in state s' and action of termination p, performed to current state s', the next state s'' of data resource i is not possible:

$$\forall i \forall p \forall s'[[I(i) \wedge P(p) \wedge S(s) \wedge delete(p, i, s')] \Rightarrow \neg \exists s''[S(s) \wedge next\_state(i, s'', p)]].$$

Action of Modification

An action of modification is a specific case of action of transference. An action of modification does not change the state of data resource: current and next state of data resource according to action of modification is the same. It offends dependency of state transition. When Enterprise modeling approach is used to specify model of information system including not only CIS then specification can be made without offences. For example, sometime data resource is in need of correction when user inputted erroneous data. Correction can be made to the data resource in the certain state only. The user of CIS must indicate states of data resource to which the corrections can be made because not in each state data resource can be corrected. Action of correction is analogous to the action of transference. When action p transfers (or creates) data resource i to (in) the state s, in which action of correction is allowed (see Fig. 6), it transfers this data resource to the state "Can be estimated", which is a specialization of state s.

User of CIS can estimate data resource being in state "Can be estimated" and classify it as correct or not correct. Action of estimation transfers data resource to the state "Estimated". This action is an action of classification also: after estimation data resource

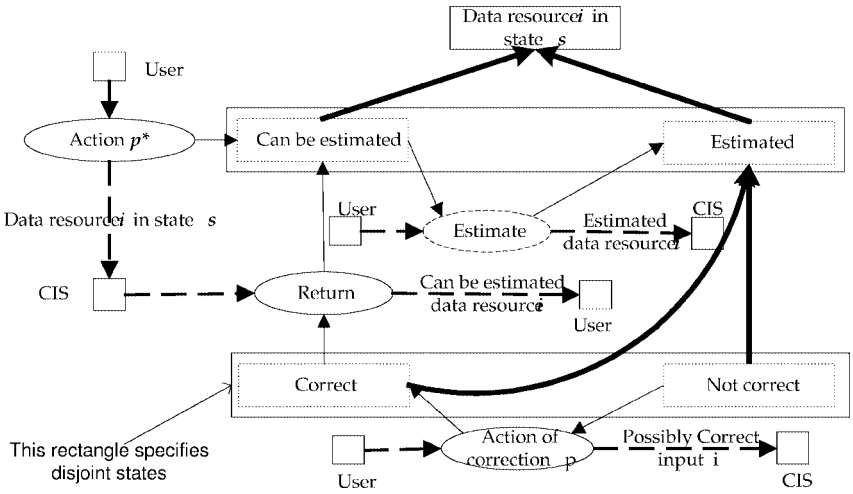

Fig. 6. Model of process of correction.

transits to the states "Correct" or "Not correct". States "Correct" or "Not correct" is a specialization of state "Estimated". If data resource is estimated as "Not correct" user can perform action of correction and transfer data resource to the state "Correct". The user of CIC is human, so, estimation of data resource is subjective: data resource estimated as "Correct" really can be not correct, and data resource estimated as "Not correct" really can be correct. User of CIS cannot ensure that data inputted by himself/herself are not erroneous. Even, after performance of action of corrections he/she cannot be sure that no error was made. Therefore, correct (possibly) data resource is returned to the state "Can be estimated" (because repeated correction decreases probability of errors). The action of estimation is not computerized in most cases because it is impossible (or very difficult) to compare and estimate data presented in computerized and not computerized form. But, action of correction is an action of additional service, which makes the CIS more flexible. So, realization of them is desirable. However, actions of estimation and correction are not business actions in the sense analyzed by us. So, computerization of states of data resource i "Can be estimated", "Estimated", "Correct", and "Not correct" is not necessary. In this case, model of process of correction becomes simpler (see Fig. 7).

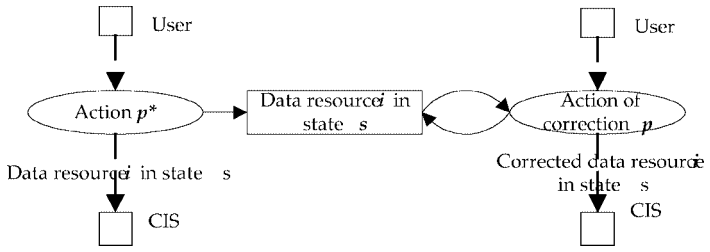

Fig. 7. Simpler model of process of correction.

In this model action of correction breaks dependency of state transition (from the viewpoint of CIS after performance of action of correction data resource do not change the state). But it is valid because action of correction changes values of the attributes, which change the state of data resource. Action of correction is a case of action of modification. Another case of application of action of modification is discussed below.

Two types of dependencies (bijection and surjection) (Matuzevičius, 1982) specified among data resources will be used in the formulas presented below. Bijection dependency can be defined as follows:

– A and B are in bijection dependency (or A and B is in binary association of (1,1;1,1) kind) if each instance of A is dependent on exactly one instance of B, and each instance of B is dependent on exactly one instance of A.

Surjection dependency can be defined as follows:

– A and B are in surjection dependency (or A and B is in binary association of (1,*;1,1) kind) if each instance of A is dependent on one or more instances of B, and each instance of B is dependent on exactly one instance of A.

Graphical notation of bijection and surjection in Martin/Odell's style (Martin and Odell, 1996) are presented in Fig. 8.

If action $\boldsymbol{p}$ transfers (creates) data resource $\boldsymbol{i''}$ to the state $\boldsymbol{s''}$, in which data resource $\boldsymbol{i''}$ is in surjection dependency with another data resource $\boldsymbol{i'}$ being in certain state $\boldsymbol{s'}$, then action $\boldsymbol{p}$ modifies data resource $\boldsymbol{i'}$ being in state $\boldsymbol{s'}$ (see Fig. 8). When action $\boldsymbol{p}$ is not computerized then this action $\boldsymbol{p}$ transfer data resource $\boldsymbol{i'}$ from state $\boldsymbol{s^*}$ to state $\boldsymbol{s^{**}}$. An action $\boldsymbol{p^*}$ is needed to return data resource $\boldsymbol{i'}$ to the state $\boldsymbol{s^*}$ if repeated performance of action $\boldsymbol{p}$ is allowed.

When action $\boldsymbol{p}$ is computerized, and different users perform actions $\boldsymbol{p}$ and $\boldsymbol{p^*}$ then the model presented in Fig. 11 is analogical to the previous model (see Fig. 9).

If action $\boldsymbol{p}$ is computerized, and the same user is obligated to perform action $\boldsymbol{p}$ and action $\boldsymbol{p^*}$, and states $\boldsymbol{s^*}$ and $\boldsymbol{s^{**}}$ are not used to form any result, and these states do not make influence to any other action, then action $\boldsymbol{p}$ is an action of modification, which modifies data resource $\boldsymbol{i'}$ being in state $\boldsymbol{s'}$, actions $\boldsymbol{p^*}$ is merged with action $\boldsymbol{p}$, and specification of states $\boldsymbol{s^*}$ and $\boldsymbol{s^{**}}$ is not needed (see Fig. 11).

From the viewpoint of the CIS after performance of action $\boldsymbol{p}$ data resource $\boldsymbol{i'}$ stays in the same state $\boldsymbol{s'}$.

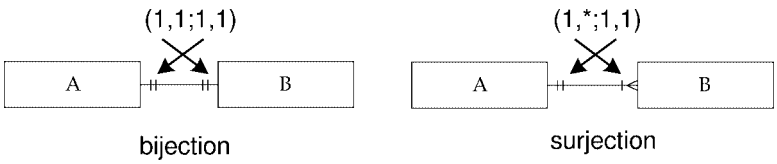

Fig. 8. Graphical notation of bijection and surjection.

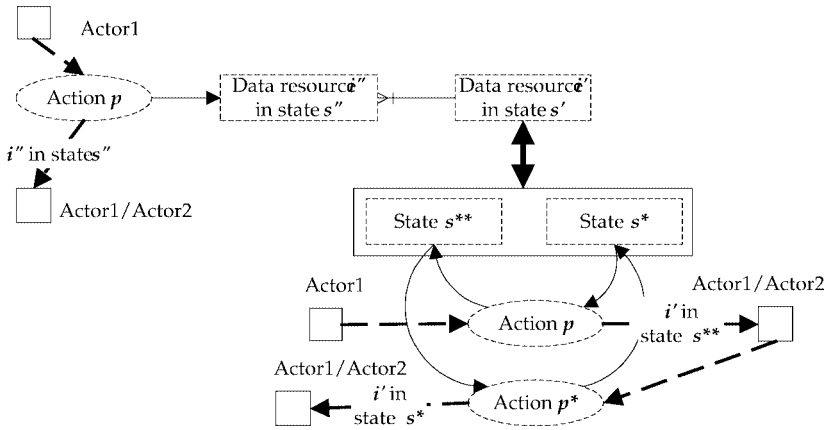

Fig. 9. Not computerized modification of input specified as actions of transference.

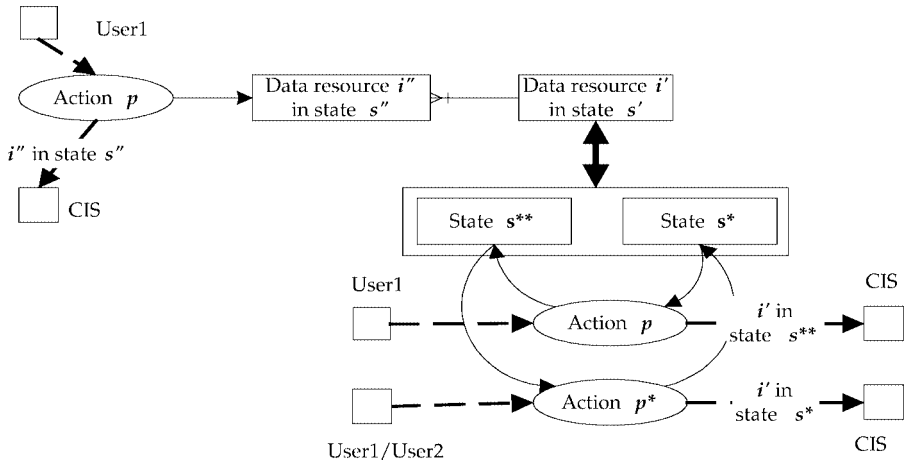

Fig. 10. Computerized modification of input specified as actions of transference.

If each action p modifies data resource i being in state s then current (and next) state of data resource i before (and after) performance of action p is state s:

$$\forall p \forall i \forall s[P(p) \wedge I(i) \ \wedge \ S(s) modify(p,i,s)] \Rightarrow [current\_state(i,s,p) \\ \wedge \ next\_state(i,s,p)]].$$

Two cases of action of modification can be identified: 1) the history of changes made to data resource is not stored, and 2) the history of changes made to data resource is stored.

Only the user can say which of mentioned above cases to use for certain data resource in certain state.

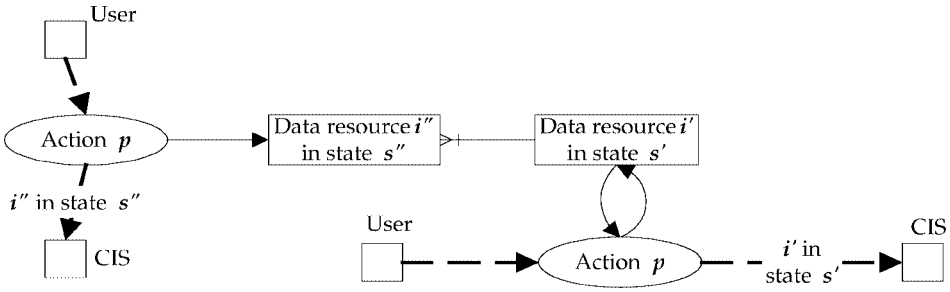

Fig. 11. Model of computerized action of modification.

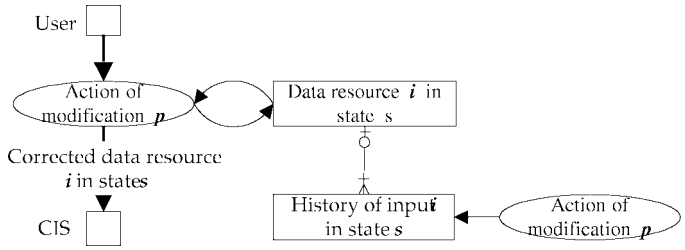

Fig. 12. The action of modification with history saving.

If user of CIS decides to store the history of changes made to the data resource i being in state s after performance of action p then new static object h to store the history of changes must be specified also (see Fig. 12). The object h is an aggregate of attributes, which are specified to describe the state s of data resource i, and which can be modified.

The total functional dependency (1,*;0,1) must be specified among data resource i in state s and object h. It means that data resource i in state s can be modified recurrently or can never be modified, and object h depends on data resource i in state s. An action of modification p, which is specified to modify the data resource i in state s, creates new item of object h:

$$\forall i \forall s \forall p[[I(i) \wedge S(s) \wedge P(p) \wedge modify(p,i,s) \wedge correct\_and\_store(p,i,s)] \Rightarrow \exists h[H(h) \wedge total\_functional(i,s,h) \wedge create(p,h)]].$$

4. Verification Rules to Estimate Dynamic and Static Dependencies Specified Among Data Resources

The objective of verification is to check whether the specification is consistent, and whether it correctly expresses the functional requirements stated by user. The verification problem can be viewed at the syntactical and semantical levels. At the syntactical level the problem is to prove that the specification is complete and free from contradictions. At the semantical level we have to take into account the meaning of terms given

in the specification, and to look for inconsistencies. Rules presented in this paper are for verification of specification at syntactical level.

Rules presented in previous chapters define haw to specify certain objects (i.e., output reports, data resources, states, actions, etc.) not estimating dependencies specified among different data resources. It is necessary to note that the change of state of certain data resource can change the state of other data resource if these data resources are associated. The action specified for the data resource being in a certain state, which is associated with other data resource, can be complex. Rules presented below help in finding of faults made in specification estimating dynamic and static dependencies specified among data resources.

Rules presented below are proper if states of data resource are disjoint.

Rule 1. If state s' is a current state of data resource i' before performance of action p, and data resource i' in state s' is in association r' kind of (1,1;1,1) or in at least one association r'' kind of (1,*;1,1) with data resource i'' in state s'' and action p according to state s' is not an action of modification then state s'' is a current state of data resource i'' before performance of action p:

$$\forall i'\forall i''\forall s'\forall s''\forall r'\forall r''\forall p[[I(i) \wedge S(s) \wedge R_S(r) \wedge P(p) \wedge$$
$$current\_state(i', s', p) \wedge \neg modify(p, i', s') \wedge [[relate(r', i', s', i'', s'')$$
$$\wedge bijection(r')] \vee [relate(r'', i', s', i'', s'') \wedge surjection(r'')]]]$$
$$\Rightarrow current\_state(i'', s'', p)].$$

The example of the situation, which illustrates the application of Rule 1, is depicted in the Fig. 13. Designed product was assembled. "Designed" (s') is a current state of product (i') before action "Assemble" (p). Parts will be used in production of designed product are selected (i'' in state s''), i.e., designed product (i' in state s') is in surjection dependency (r) with part for designed product (i'' in state s''). So, the current state of this part before action "Assemble" is "Part for designed product".

Rule 2. If state s' is a next state of data resource i' after performance of action p, and data resource i' in this state is in at least one association r' kind of (1,1;1,1) or in at least one association r'' kind of (1,*;1,1) with data resource i'' being in state s'' and action p according to state s' is not an action of modification then state s'' is a next state of data resource i'' after performance of action p:

$$\forall i'\forall i''\forall s'\forall s''\forall r'\forall r''\forall p[[I(i) \wedge S(s) \wedge R_S(r) \wedge P(p) \wedge next\_state(i', s', p)$$
$$\wedge \neg modify(p, i', s') \wedge [[relate(r', i', s', i'', s'') \wedge bijection(r')]$$
$$\vee [relate(r'', i', s', i'', s'') \wedge surjection(r'')]]] \Rightarrow next\_state(i'', s'', p)].$$

The example of the situation, which illustrates the application of Rule 2, is depicted in the Fig. 13. Designed product was assembled. "Assembled" (s') is a next state of product (i') after action "Assemble" (p). Assembled product does not exist without its parts (i''

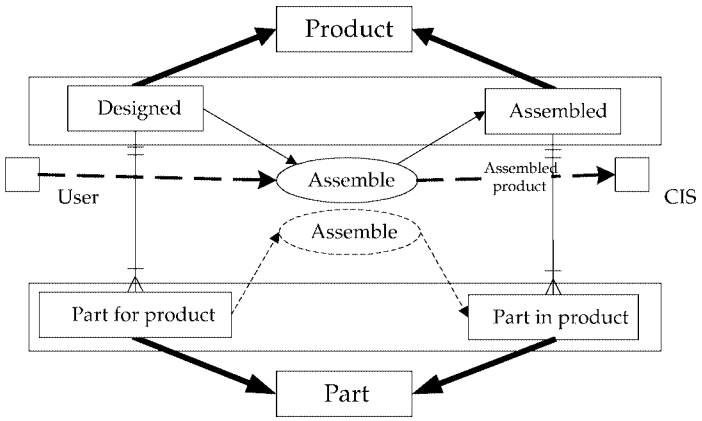

Fig. 13. Model of production of product.

in state s''), i.e., produced product (i' in state s') is surjection dependency (r) with part in produced product (i'' in state s''). So, the next state of part after action "Assemble" is "Part of product".

Rule 3. If state s'' is a current state of data resource i'' before performance of action p, and data resource i'' in this state is in at least one association r kind of (1,1;1,*) with data resource i' being in state s' then state s' is just a current state of data resource i' before performance of action p, or state s' is a state, which is modified by action p:

$$\forall i'\forall i''\forall s'\forall s''\forall r\forall p[[I(i) \wedge S(s) \wedge R_S(r) \wedge P(p) \wedge current\_state(i'', s'', p) \wedge relate(r, i', s', i'', s'') \wedge surjection(r)] \Rightarrow [current\_state(i', s', p) \vee modify(p, i', s')]].$$

For example, if parts for designed product are used in assemblage (see Fig. 13) then designed product is involved in this action also. If part of designed product can be rejected (see Fig. 14) then after performance of action of rejection the designed product, which part was rejected, is modified (action "Reject" is action of modification with respect to the designed product, and action of termination with respect to the part of designed product).

Rule 4. If state s'' is a next state of data resource i'' after performance of action p, and data resource i'' in this state is in at least one association r kind of (1,1;1,*) with data resource i' being in state s' then state s' is just a next state of data resource i' after performance of action p, or state s' is a state, in which data resource i' is modified by action p:

$$\forall i'\forall i''\forall s'\forall s''\forall r\forall p[[I(i) \wedge S(s) \wedge R_S(r) \wedge P(p) \wedge next\_state(i'', s'', p) \wedge relate(r, i', s', i'', s'') \wedge surjection(r)] \Rightarrow [next\_state(i', s', p) \vee modify(p, i', s')]].$$

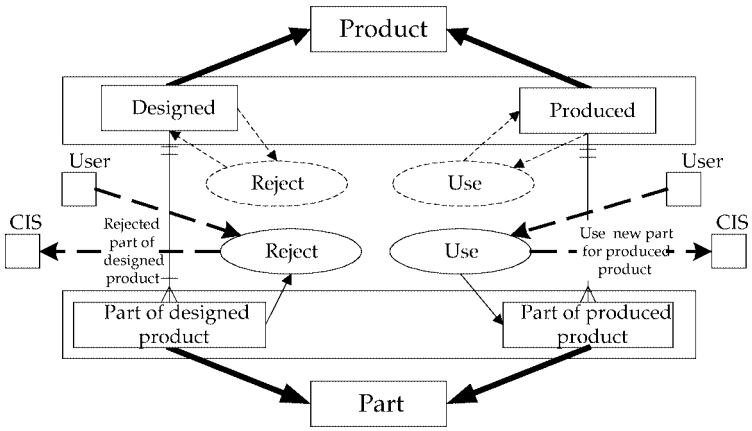

Fig. 14. Model of rejection of part of product and the usage of part of product.

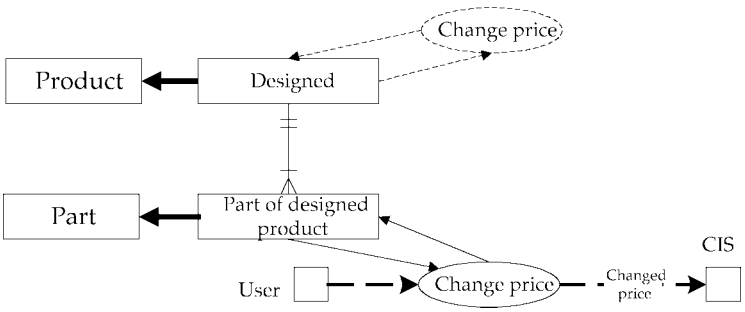

Fig. 15. Model of change of price of part of product.

For example, if parts in product designed were used in assemblage (see Fig. 13) then assembled product was involved in assemblage also. If new part was used to produce product (see Fig. 14) then after performance of action of use the produced product (for which part was used) is modified (action "Use" is action of modification with respect to the produced product, and action of creation with respect to the part of produced product).

Rule 5. If state s'' of data resource i'' can be modified by action p, and data resource i' in state s' is in at least one association r' of (1,1;1,1) kind or in at least one association r'' of (1,*;1,1) kind with data resource i'' being in state s'', then data resource i' in state s' is modified by action p also:

$$\forall i'\forall i\forall s'\forall s''\forall r'\forall r''\forall p[[''I(i) \wedge S(s) \wedge R_S(r) \wedge P(p) \wedge$$
$$modify(p, i'', s'') \wedge [[relate(r', i', s', i'', s'') \wedge bijection(r')] \vee$$
$$[relate(r'', i', s', i'', s'') \wedge surjection(r'')]]] \Rightarrow modify(p, i', s')].$$

For example (see Fig. 15), if price of part of designed product was changed then price of product is changed also.

Rule 6. If state s' is a next state of data resource i' after performance of action p, and state s'' is a current state of data resource i'' before performance of action p then state s^* of data resource i' exist, and this state is a current state of data resource i' before performance of action p, and/or state s^{**} of data resource i'' exist, and this state is a next state of data resource i'' after performance of action p:

$$\forall i'\forall i''\forall s'\forall s''\forall p[[I(i') \wedge I(i'') \wedge S(s') \wedge S(s'') \wedge P(p) \wedge next\_state(i', s', p) \wedge current\_state(i'', s'', p)] \Rightarrow \exists s^*[S(s^*) \wedge current\_state(i', s^*, p)] \vee \exists s^{**}[S(s^{**}) \wedge next\_state(i'', s^{**}, p)]].$$

For example (see Fig. 13), if "Assembled" is a next state of product after performance of action "Assemble", and if "Part for product" is a current state of part before performance of action "Assemble" then current state of product (for example "Designed") and/or next state of part (for example "Part in product") must be specified.

Rule 7. If data resource i' in state s' is in at least one association r of (1,*;1,1) kind with data resource i'' being in state s'', and data resource i' being in state s' can be modified by action p, and state s'' is a next state of data resource i'' after performance of action p, and action p is not an action of modification according to data resource i'' being in state s'' then action p can not be applied for first data resource i'' being in state s'' and relating data resource i' in state s':

$$\forall i'\forall i''\forall s'\forall s''\forall r\forall p[[I(i') \wedge I(i'') \wedge S(s') \wedge S(s'') \wedge R_S(r) \wedge P(p) \wedge relate(r, i', s', i'', s'') \wedge surjection(r) \wedge modify(p, i', s') \wedge next\_state(i'', s'', p) \wedge \neg modify(p, i'', s'')] \Rightarrow not\_for\_first(p, i', s', i'', s'')].$$

For example (see Fig. 14), produced product can have at least one part. If new part is used to produce product ("Part of produced product" is a next state of part after performance of action of use), and action of use modifies produced product and do not modifies part of produced product then action of use is not applied to usage of the first part of certain produced product.

Rule 8. If data resource i' in state s' is in at least one association r of (1,*;1,1) kind with data resource i'' being in state s'' and data resource i' being in state s' can be modified by action p and state s'' is a current state of data resource i'' before performance of action p, and action p is not an action of modification according to data resource i'' being in state s'' then action p can not be applied for last data resource i'' being in state s'' and relating with data resource i' in state s':

$$\forall i'\forall i''\forall s'\forall s''\forall r\forall p[[I(i') \wedge I(i'') \wedge S(s') \wedge S(s'') \wedge R_S(r) \wedge P(p) \wedge relate(r, i', s', i'', s'') \wedge surjection(r) \wedge modify(p, i', s') \wedge current\_state(i'', s'', p) \wedge \neg modify(p, i'', s'')] \Rightarrow not\_for\_last(p, i', s', i'', s'')].$$

For example (see Fig. 14), designed product can have at least one part. If part of designed product can be rejected ("Part of designed product" is a current state of part before

performance of action of rejection), and action of rejection modifies designed product and do not modifies part of designed product then action of rejection is not applied to reject the last part of certain designed product.

Rule 9. If state s' is a next state of data resource i' after performance of action p, and state s'' is a next state of data resource i'' after performance of action p, or if state s' is a current state of data resource i' before performance of action p, and state s'' is a current state of data resource i'' before performance of action p then at least one association r between data resource i' being in state s' and data resource i'' being in state s'' exist:

$$\forall i' \forall i'' \forall s' \forall s'' \forall p[[I(i') \wedge I(i'') \wedge S(s') \wedge S(s'') \wedge P(p) \wedge [[next\_state(i', s', p) \wedge next\_state(i'', s'', p)] \vee [current\_state(i', s', p) \wedge current\_state(i'', s'', p)]] \Rightarrow \exists r[R_S(r) \wedge relate(r, i', s', i'', s'')]].$$

For example (see Fig. 13), "Designed product" and "Part for product" are current states of action "Assemble". So, association between data resources in these states must exist (in this case a dependency of surjection is specified). "Assembled product" and "Part in product" are next states of action "Assemble". So, association between data resources in these states must be specified (in this case a dependency of surjection is specified).

Rule 10. If state s^* of data resource i is related with other state s' of data resource i by IS_A dependency $z(s^* \Rightarrow s')$ and state s' is current state of data resource i according to action p, and action p is not an action of modification, then state s^* is current state of data resource i according to action p also:

$$\forall i \forall p \forall s^* \forall s' \forall z[[I(i) \wedge P(p) \wedge S(s^*) \wedge S(s') \wedge Z(z) \wedge is\_a(z, i, s^*, s') \wedge current\_state(i, s', p) \wedge \neg modify(p, i, s')] \Rightarrow current\_state(i, s^*, p)].$$

State "Drafted" is a current state (s') of data resource "Product" (i) according to action "Reject drafted product" (see Fig. 16). State "Designed" (s^*) is a specialization of data resource "Product" in state "Drafted". So, state "Designed" is a current state of data resource "Product" according to action "Reject drafted product". State "Not designed"

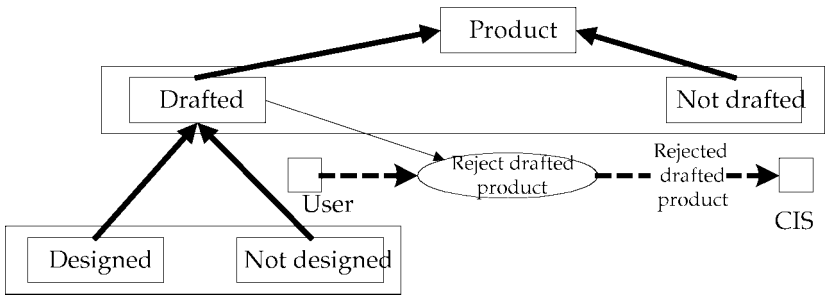

Fig. 16. Model of rejection of product.

(s^*) is a specialization of data resource "Product" in state "Drafted" also. So, state "Not designed" is a current state of data resource "Product" according to action "Reject drafted product". In other words, if user can reject drafted product he/she can reject designed or not designed product also (depends on in which state ("Designed" or "Not designed") drafted product is at the moment of rejection).

Rule 11. If state s^* of data resource i is related with other state s' of data resource i by IS_A dependency $z(s^* \Rightarrow s')$ and state s' is next state of data resource i according to action p, and action p is not an action of modification, then state s^* is next state of data resource i according to action p also:

$$\forall i \forall p \forall s^* \forall s' \forall z [[I(i) \wedge P(p) \wedge S(s^*) \wedge S(s') \wedge Z(z) \wedge is\_a(z, i, s^*, s') \wedge next\_state(i, s', p) \wedge \neg modify(p, i, s')] \Rightarrow next\_state(i, s^*, p)].$$

State "Paid" is a next state (s') of data resource "Invoice" (i) according to action "Pay" (see Fig. 17). State "Fully" (s^*) is a specialization of data resource "Invoice" in state "Paid". So, state "Fully" is a next state of data resource "Invoice" according to action "Pay". State "Partially" (s^*) is a specialization of data resource "Invoice" in state "Paid" also. So, state "Partially" is a next state of data resource "Invoice" according to action "Pay". In a other words, if user can pay not paid invoice he/she can pay invoice fully or partially (amount of payment describe to which state invoice will get).

For the description of the rules presented below two sub-sets of data resources can be defined according to the number of states of data resources:

$I = I' \cup I''$, where

I' – subset of data resources having more than one state,

I'' – subset of data resources having one state only,

$$\forall i I'(i) \Rightarrow \exists s \exists s^* [S(s) \wedge state\_of\_data\_resources(i, s) \wedge state\_of\_data\_resources(i, s^*)],$$
$$\forall i I''(i) \Rightarrow \exists! s [S(s) \wedge state\_of\_data\_resources(i, s)].$$

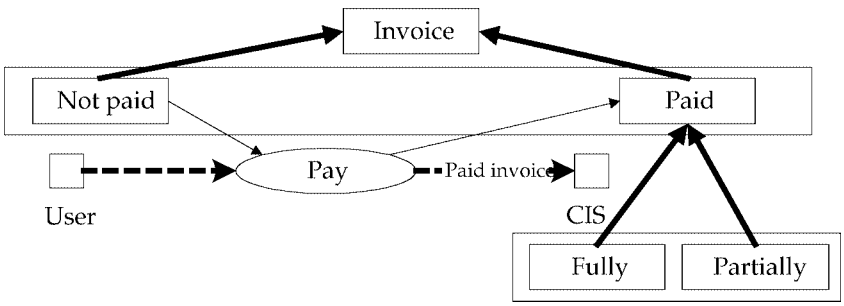

Fig. 17. Model of payment.

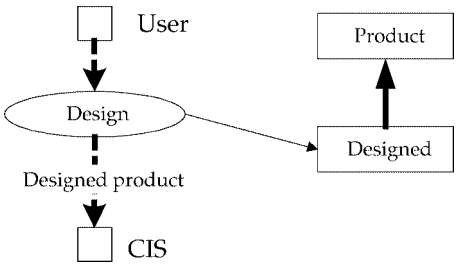

Fig. 18. Model of design of product.

Rule 12. If state s is a next state of data resource i after performance of action p and state s is only state of data resource i then action p creates data resource i in state s:

$$\forall i \forall s \forall p[[I''(i) \wedge S(s) \wedge P(p) \wedge next\_state(i, s, p)] \Rightarrow create(p, i, s)].$$

The example of the situation, which illustrates the application of Rule 12, is depicted in the Fig. 18. Product was designed. State "Designed" (s) is a next state of data resource "Product" (i) after action "Design" (p). Only state "Designed" of data resource "Product" is specified, i.e., there are no other states of data resource "Product" specified, and this specification is right (there are no requirements to specify more states). So the action "Design" is the action of creation of data resource "Product" (i) in state "Designed" (s).

Rule 13. If state s is a current state of data resource i before performance of action p and state s is only state of data resource i then action p deletes data resource i in state s:

$$\forall i \forall s \forall p[[I''(i) \wedge S(s) \wedge P(p) \wedge current\_state(i, s, p)] \Rightarrow delete(p, i, s)].$$

The example of the situation, which illustrates the application of *Rule 13*, is depicted in the Fig. 19. Designed product was rejected. State "Designed" (s) is a current state of data resource "Product" (i) before action "Reject" (p). Only state "Designed" of data resource "Product" is specified, i.e., there are no other states of data resource "Product" specified, and this specification is right (there are no requirements to specify more states). So the action "Reject" is the action of deletion of data resource "Product" (i) being in state "Designed" (s).

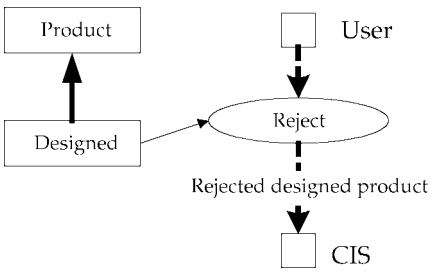

Fig. 19. Model of rejection of product.

Rule 14. If state s'' is a next state of data resource i after performance of action p and state s'' is not only state of data resource i, then action p creates data resource i in state s'' or transfers state of data resource i from state s' to state s'':

$$\forall i \forall s'' \forall p[[I'(i) \wedge S(s'') \wedge P(p) \wedge next\_state(i, s'', p)] \Rightarrow [create(p, i, s'') \wedge \neg\exists s'[S(s') \wedge transfer(p, i, s', s'')] \wedge \neg modify(p, i, s'')] \vee [\neg create(p, i, s'') \wedge \exists s'[S(s') \wedge transfer(p, i, s', s'')]] \wedge \neg modify(p, i, s'')] \vee [\neg create(p, i, s'') \wedge \neg\exists s'[S(s') \wedge transfer(p, i, s', s'')] \wedge modify(p, i, s'')]].$$

If data resource has more than one state there are no verification rule, which gives only correct solution. The *Rule 14* gives two possible solutions: kind of action p must be "create" or "transfer". If problem domain is such that it is possible to reach state s'' of data resource i in two ways, then two different actions must be specified.

For example see Fig. 20. State "Designed" (s'') is the next state of data resource "Product" (i) after action "Design". Kind of action "Design" can be "create" or "transfer". If kind of action "Design" is "create" then this action creates new data resource "Product" in state "Designed", and it means that for the design of the new product no preconditions are required (for example draft of product). If kind of action "Design" is "transfer" then: 1) one more state s' (in this case state "Drafted") of data resource "Product" exists, 2) this state is a current state of data resource "Product" before action "Design", 3) action "Design" transfers state of data resource "Product" from state "Drafted" to state "Designed". It means that for the design of product the draft of this product is required. If problem domain is such that it is possible to get a design of product in two ways, then two different actions of design must be specified: "Design without draft" and "Design from draft". Only user who knows problem domain can explain the ambiguous situations.

Rule 15. If state s' is a current state of data resource i before performance of action p, and state s' is not the only state of data resource i, then action p deletes data resource i in state s', or transfers state of data resource i from state s' to state s'':

$$\forall i \forall s' \forall p[[I'(i) \wedge S(s) \wedge P(p) \wedge current\_state(i, s', p)] \Rightarrow [delete(p, i, s') \wedge \neg\exists s''[S(s'') \wedge transfer(p, i, s', s'')]] \wedge \neg modify(p, i, s'')] \vee$$

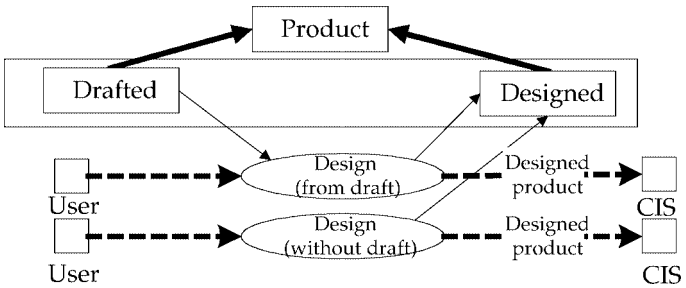

Fig. 20. Model of two cases of design of product.

$$[\neg delete(p,i,s') \wedge \exists s''[S(s'') \wedge transfer(p,i,s',s'')] \wedge \neg modify(p,i,s'')] \vee$$
$$[\neg delete(p,i,s') \wedge \neg\exists s''[S(s'') \wedge transfer(p,i,s',s'')] \wedge modify(p,i,s'')]].$$

If data resource has more than one state there are no verification rule, which gives only one correct solution. The Rule 15 gives two possible solutions: kind of action $\boldsymbol{p}$ must be "delete" or "transfer". If problem domain allows exit from state $\boldsymbol{s'}$ of data resource $\boldsymbol{i}$ in two ways, then two different actions must be specified. For example see Fig. 21.

State "Designed" ($\boldsymbol{s'}$) is the current state of data resource "Product" ($\boldsymbol{i}$) before action "Reject". Kind of action "Reject" can be "delete" or "transfer". If kind of action "Reject" is "delete", then this action deletes data resource "Product" being in state "Designed". It means that designed product can be rejected as not useful, and all information about it is deleted. If kind of action "Reject" is "transfer" then one more state s" (in this case state "Drafted") of data resource "Product" exists. This state is a next state of data resource "Product" after action "Reject", and action "Reject" transfers state of data resource "Product" from state "Designed" to state "Drafted". It means that if only design of product is rejected, then the product becomes not designed. If it is allowed in a problem domain to behave in two ways with a designed product, then two different "Reject" actions must be specified: "Reject design" and "Reject product". Only user who knows problem domain can explain such ambiguous situation.

More rules can be defined if states of data resource would not be disjoint, or if static dependencies would be weaker. If states are not disjoint and have associations with states of other data resource, then some weak static dependencies can be inferred. Weak binary associations are full of semantic holes. Semantic holes create ambiguity of actions, so, model with not disjointed states and weak associations must be re-modeled to delete ambiguous situations. Not disjoint states must be re-modeled in order to become disjoint by including new states or by accepting precondition that specified states are disjoint because new states are not relevant to CIS. Semantic holes can be deleted by conceptual operations of prefixing, restriction, association, grouping and negation (Gustas, 1994).

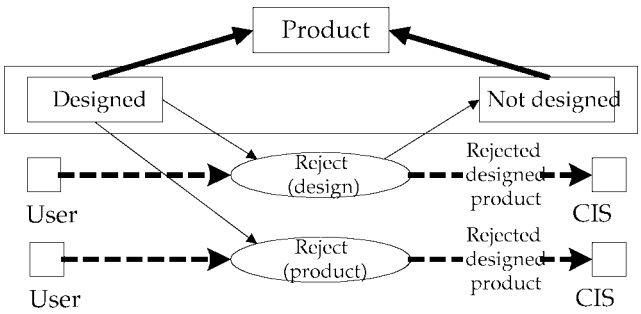

Fig. 21. Model of two cases of rejection of designed product.

5. Verification Rules for Identification of Missing and not Relevant to CIS Data Resources, States, and Actions

When specification of data resources, states of data resources and actions, which process the data resources, are specified such kind of questions arise:

1. Are all the data resources of CIS specified?
2. Are all the specified data resources relevant to CIS?
3. Are all the states of data resources of CIS specified?
4. Are all the specified states of data resources relevant to CIS?
5. Are all the actions, which process the data resources of CIS, specified?
6. Are all the specified actions, which process the data resources, relevant to CIS?

Answers to the questions:

1. Yes, if it is possible to form all output reports of CIS from the specified data resources, i.e., for each output report $\boldsymbol{o}$ and attribute $\boldsymbol{t'}$, which describes output report $\boldsymbol{o}$, data resource $\boldsymbol{i}$ and attribute $\boldsymbol{t''}$, which describes data resource $\boldsymbol{i}$, must be specified which is used to form the attribute $\boldsymbol{t'}$ of output report $\boldsymbol{o}$:

$$\forall o \forall t'[O(o) \wedge T(t') \wedge describe(o, t') \Rightarrow \exists i \exists t''[I(i) \wedge T(t'') \wedge describe(i, t'') \wedge use\_attribute(o, t', i, t'']].$$

   Otherwise:

   a) A specific data resource is not specified, or
   b) A specific state of certain data resource is not specified.

2. Yes, if there is no data resource, which is not used to form any output report of CIS. I.e., for each data resource $\boldsymbol{i}$ attribute $\boldsymbol{t'}$ must specified which describes data resource $\boldsymbol{i}$, and is used to form the attribute $\boldsymbol{t''}$, which describes output report $\boldsymbol{o}$:

$$\forall i[I(i) \Rightarrow \exists t' \exists o \exists t''[T(t') \wedge describe(i, t') \wedge O(o) \wedge T(t'') \wedge describe(o, t'') \wedge use\_attribute(o, t'', i, t')]].$$

   Data resource, which is not used to form any output report of CIS, is redundant, or the user of CIS has forgotten to indicate the output report, for which formation the specified data resource is used. In latter case, the output report must be specified and all specification of CIS must be reviewed and made more accurate. But, if user does not confirm the need of specified data resource, this redundant data resource must be deleted from the specification of CIS.

3. Yes, if it is possible to form all output reports of CIS from the specified states of data resources, i.e., for each output report $\boldsymbol{o}$ and attribute $\boldsymbol{t'}$, which describes output

report o, attribute t'' must be specified which describes data resource i being in state s, and is used to form the attribute t'' of output report o:

$$\forall o \forall t'[O(o) \wedge T(t') \wedge describe(o, t') \Rightarrow \exists t'' \exists i \exists s[T(t'') \wedge I(i) \wedge S(s) \wedge describe\_state(i, s, t'') \wedge use\_attribute(o, t', i, t'']].$$

No, if all data resources (used to form output reports of CIS) are specified, but there is not enough of specified states of data resources to form all output reports of CIS.

4. Yes, if there is no such state of data resource, which is not used to form at least one output report of CIS, i. e., for each data resource i being in state s attribute t' must specified which describes data resource i being in state s, and is used to form the attribute t'', which describes output report o:

$$\forall i \forall s[I(i) \wedge S(s) \wedge state\_of\_data\_resource(i, s)] \Rightarrow \exists t' \exists o \exists t''[T(t') \wedge describe\_state(i, s, t') \wedge O(o) \wedge T(t'') \wedge describe(o, t'') \wedge use\_attribute(o, t'', i, t')]].$$

State of data resource, which is not used to form any output report of CIS, is redundant, or the user of CIS has forgotten to indicate the output report, for which formation the specified state of data resource is used. In latter case, such output report must be specified, and all specification of CIS must be revised and made more accurate. But, if user does not confirm the need of specified state of data resource, this redundant state of data resource must be deleted from specification of CIS, and the specification of CIS must be revised, also.

5. Yes, if it is possible to reach all specified states of data resources of CIS, i.e., for each state s of input i at least one action p must be specified which transfer (create) input i to (into) state s (state s is a next state respect to the action p):

$$\forall i \forall s[[I(i) \wedge S(s) \wedge state\_of\_data\_resource(i, s)] \Rightarrow \exists p[P(p) \wedge next\_state(i, s, p)]].$$

But if a certain state of data resource cannot be reached, it means that specific action is not specified. If not reachable state is relevant to CIS then the missing action must be specified. Otherwise, not reachable state must be deleted from the specification of CIS. Specifications of CIS must be revised after each change.

6. Yes, if there is no action, which transfer the data resource of CIS to a state not relevant to CIS, i.e., for each action p, which transfer (create) data resource i to (into) state s, attribute t' must be specified which describe data resources i being in state s and is used to form attribute t'' of output report o:

$$\forall p \forall i \forall s[P(p) \wedge I(i) \wedge S(s) \wedge next\_state(i, s, p)] \Rightarrow \exists t' \exists o \exists t''[T(t') \wedge describe\_state(i, s, t') \wedge O(o) \wedge T(t'') \wedge describe(o, t'') \wedge use\_attribute(o, t'', i', t')]].$$

If such action is specified and if user confirms not relevance of state of data resource (see answer 4), then this action and corresponding not relevant state (to which this action transfers) must be deleted from the specification of CIS, and specification of CIS must be revised.

Deleted specified states of the data resources and actions, which transfer data resources to these deleted states, are possibly needful to achieve the goals of the organization, but they are not needful to realize functions of CIS (to serve the purpose of the CIS). All actions (specified and not specified) are performed in organization, but actions performed by CIS must be specified. CIS purpose (which is based on the output reports) determines what kinds of data resources, states, and actions must be specified.

6. Conclusion

It is well-known fact that deficiencies introduced in the early phases of systems development are the most difficult and costly ones to correct, unless detected early. Rules presented in the paper can be applied for detection of certain shortages in specification of data resources of CIS and actions processing them.

Qualities of this research can be defined as follows:

- classification of actions gives an opportunity to describe the life cycle of data resources;
- rules presented take into account static and dynamic dependencies;
- rules presented check the specified data resources and actions to achieve the specified purpose of CIS.

Verification rules presented in this paper can be implemented in CASE-tool to facilitate the specification of computerized information system.

The future work is planned on evolution of the model for the user requirement specification, and on preparation of verification rules in the later stages of CIS development (for instance, development of entities and relationships among them from the specified data resources).

Acknowledgement

The authors are very grateful to the anonymous Referee that has helped to improve the previous versions of this paper.

References

Barker, R. (1990). *CASE*METHOD: Tasks and Deliverables,* Addison – Wesley Publ. Co., New York.

Barker, R., C. Longman (1992). *CASE*METHOD: Function and Process Modelling,* Addison – Wesley Publ. Co., New York.

Booch, G. (1994). *Object Oriented Analysis and Design with Applications*. 2nd ed. Benjamin/Cummings, Redwood City, CA.

Butkienė, R., R. Butleris (1997). Ekspertinio informacijos sistemų projektavimo kompiuterizavimas, *Informacinės Technologijos'97*, Technologija, Kaunas, 3–6 (in Lithuanian).

Butkienė, R., R. Butleris (1998). A Framework for the user requirements specification. *Informacinės Technologijos ir Valdymas*, **2**(8), 40–53.

Coad, P., E. Yourdan (1991). *Object-Oriented Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.

Cauvet, C., C. Proix, C. Rolland (1991). ALECSI: An expert system for requirements engineering. In *Proceedings of Third International Conference CAiSE'91*. Trondhein, Norway, pp. 31–49.

Gustas, R. (1994). *Towards Uderstanding and Formal Definition of Conceptual Constraints, Information Modelling and Knowledge Bases VI*, IOS Press, pp. 381–399.

Gustas, R. (1997). *Semantic and Pragmatic Dependencies of Information Systems*, Monograph, Technologija, Kaunas.

Jacobson, I., M. Christerson, P. Jonnson, G. Overgaard (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Publ. Co., New York.

Loucopoulos, P. (1992). *Conceptual Modeling. Conceptual Modeling, Databases and Case: An Integrated View of Information Systems Development*, John Wiley, 1–26.

Martin, J., J.J. Odell (1995). *Object-Oriented Methods: A Foundation,* Prentice-Hall, Englewood Cliffs, New Jersey.

Matuzevičius, A (1982). *Topologija*, Mokslas, Vilnius, 228–233.

Muller, P.A. (1997). *Instant UML*. Wrox Press Ltd.

Rolland, C., C. Cauvet (1992). *Trends and Perspectives in Conceptual Modelling*. *Conceptual Modeling, Databases & Case*. John Wiley &Sons.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen (1991). *Object-Oriented Modelling and Design*, Prentice Hall Int.

R. Butkienė received the M.S. degree from Kaunas University of Technology in 1995. Currently she is Ph.D. student in the Department of Information Systems at Kaunas University of Technology. Her research interest includes requirements engineering, information systems design, and databases.

R. Butleris is an Associate Professor in information systems at Kaunas University of Technology, member of ECCAI and Computer Society of Lithuania. In 1980 he graduated from Kaunas Institute of Polytechnics (now it is Kaunas University of Technology). In 1988 he received a Ph.D. of management in technical systems from Kaunas Institute of Polytechnics. Currently his research interest includes requirements engineering, business rules specification, information system design and reengineering.

Kompiuterizuotos informacijos sistemos modelių verifikuojančios taisyklės duomenų šaltinius apdorojančių veiksmų atžvilgiu

Rita BUTKIENĖ, Rimantas BUTLERIS

Straipsnyje pateiktos taisyklės, kurių pagalba randamos klaidos kompiuterizuotų informacijos sistemų modeliuose. Kompiuterizuotos informacijos sistemos modeliui sudaryti naudojamas *Enterprise modeling* metodas, kuris integruoja statines ir dinamines priklausomybes. Šio metodo pagrindinis konstruojantysis elementas yra tipinė komunikacinių akcijų kilpa. Straipsnyje nagrinėjami tik informacijos sistemos duomenų šaltinius apdorojantys veiksmai.