Gilbert Babin and François Lustman. "Application of Formal Methods to Scenario-based Requirements Engineering." International Journal of Computers and Applications. 23(3). 2001. pp. 141-151.

APPLICATION OF FORMAL METHODS TO SCENARIO-BASED REQUIREMENTS ENGINEERING

G. Babin*

*Service d'enseignement des technologies de l'information
École des Hautes Études Commerciales
Montréal, Canada H3T 2A7
Gilbert.Babin@hec.ca

F. Lustman**

**Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, Canada H3C 3J7
lustman@iro.umontreal.ca

Abstract

The KLuB project is an attempt to use formal methods in the process and product of requirements engineering of information systems. In the work presented here, the scenario technique was used for requirements elicitation. Scenarios, which have been recognized as an effective technique for eliciting requirements, focus usually on behavior and less on data. An additional objective of the project was to integrate data and behavior in a formal specification, based on state machines. Semantic integration of data and behavior was achieved by introducing the concept of compatibility between data values and system states. Scenario integration is also achieved based on data values. An additional objective was to automate as much as possible the requirements elicitation process. The KLuB process involves three steps: the Scenario Acquisition step, the Baseline Elicitation step, and the Integration step, which is completely formal and can be automated.

Key Words

Requirements Engineering, Formal Methods, Scenarios.

1 Introduction

Scenarios have been recognized as an effective technique for eliciting requirements in general [1, 2], and for investigating behavior, in particular in the object-oriented approach [3, 4]. Scenario description, first informal, has lately been overtaken by more formal graphical approaches [3, 4]. Scenarios have also been represented by tools like relations [5] and finite state automata, used in many variations [6, 7, 8].

What does a scenario describe? Usually behavior [3, 4, 6, 9], sometimes data and behavior [5, 7, 8]. A common feature of the different definitions is that it describes only part of a system. Once scenarios are described, the next problem is to integrate them in order to obtain a system specification. In some works, the relative position of the scenarios to integrate has to be known [5, 10, 11], in others it does not [7, 8]. The integration technique is manual [5, 10], algorithmic [7, 8], or human-assisted [9, 11].

The objectives of the KLuB project are to apply formal methods in requirements engineering, to use the scenario technique, and to produce a formal specification involving data and behavior. The concept underlying the semantic integration of data and states is that of compatibility. In a scenario state, only certain data values are possible, those “in agreement” with the incoming and outgoing transitions. At the system level, states are defined by data values compatible with the same scenario states. The approach considers sequential scenarios, one instance of each data object, one instance of each scenario.

An overview of the KLuB approach is presented in Section 2. Scenario acquisition is described in Section 3. Section 4 deals with the establishment of a Baseline for scenario integration, which is the subject of Section 5. Finally in the Conclusion (Sect. 6), potential benefits, problems and possible extensions are presented.

2 Method Overview

2.1 Requirements Engineering

The KLuB process (see fig. 1) involves three steps, two informal and one formal. In the Scenario Acquisition step, which is informal, scenarios are elicited and their behavioral part is modeled into finite state machines. Additional information is required for the Integration step; the Baseline Elicitation step is intended to provide this information, namely integrated data models plus system semantics in the form of business rules. The Integration step, completely formal, generates a system specification as an extended finite state machine in which system states are based on integrated data values.

2.2 A Simple Library Example

Examples used for supporting a method are often small and allow for the “does not scale-up” criticism. To alleviate this, we decided to use a real-world system to support this work. Université de Montréal’s library system was investigated and a subset involving six scenarios was processed in [7]. However, for presentation purposes, a scaled down example, involving simplified scenarios for document borrowing, document return, and reader registration, is used here. The description of the Document

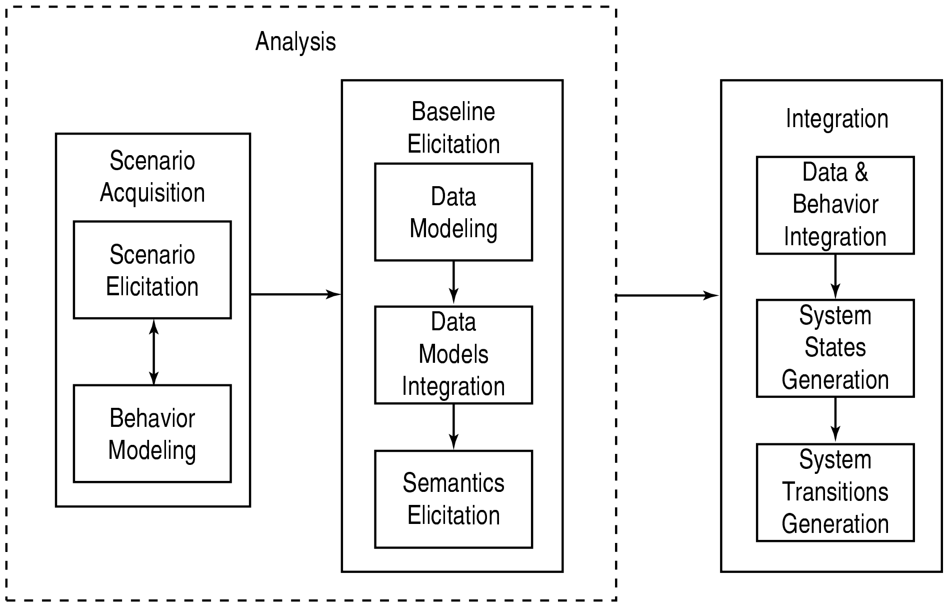

Figure 1: Requirements Engineering Process

Loan scenario is found in fig. 2.

- The user wishes that the system be in the Document Loan scenario, waiting for the reader's ID-code.
- Also, the user wants to be able to go from there directly to the reader registration scenario or to the document return scenario.
- If document borrowing should be performed, the user enters the reader's ID-code. The reader's borrowing file should appear on the screen. It should also be possible to enter the borrowing scenario at this point, from the reader registration scenario or the document return scenario.
- The user enters the document's ID-code. If the reader rights do not allow him to borrow this type of document, the loan is not processed, and the user can go back to document-ID entry state or to the initial state. If he has the right to borrow that type of document, the Loan File on the screen should be updated with that new document, and the Document's status updated from available to borrowed.
- If more than one document is to be borrowed, the user performs the preceding step as many times as required, or he (she) returns to the scenario's primary entry point, waiting for a reader's ID-code.

Figure 2: Description of the Document Loan Scenario

3 Scenario Acquisition

3.1 Definition of Scenario

There are many definitions and uses of the scenario concept [4, 5, 6, 7, 8, 10, 12]. For clarification purposes, the meaning and definition of scenario used in this work is presented. It is related to the concept of transaction introduced in [13].

Business Task A computerized information system (CIS) is to be developed as part of an information system (IS). A business task is an independent task or activity which is part of the IS. A business task has a beginning, performs an activity defined by the user, and has an ending. It leaves the IS in a coherent state in the database transaction sense.

Example: The IS is a Library system. A business task is to register a new document. The user is the clerk or librarian in charge of document registration.

Scenario A scenario is defined by a user or community of users. It defines the interactions between a single user and the CIS for performing a business task.

The basic elements of a scenario definition are one or several entry points, a set of interactions (user action, expected CIS reaction(s)), the partial or complete ordering of the interactions, and those interactions which end the scenario.

All elements contributing to the definition of a scenario are provided by or elicited from the user and are expressed in terms of the IS and the business task. This will allow for grounding in the real world of the IS the components of the formal specification given later.

3.2 Formal Scenario Model

In this work, the formal model of a scenario will be assumed to be available. The behavioral part will be modeled by a state-event automaton, a variation of a finite state machine, defined as $Mc = (Sc, Hc, Ic, Fc, Tc)$, where:

- Sc is a set of states. The scenario is in a state when a system reaction is finished and the user has not entered an event.
- Hc is the set of events. An event is a gesture exercised by the user on the interface, indicating to the system that the user has finished entering all the elements of an action (Sect. 3.1). An event is labelled by the user-defined action.
- Ic is the set of initial states. Initial states correspond to user-defined entry points.
- Fc is the set of final states. Interactions defined by the user as ending the scenario, end on a final state.

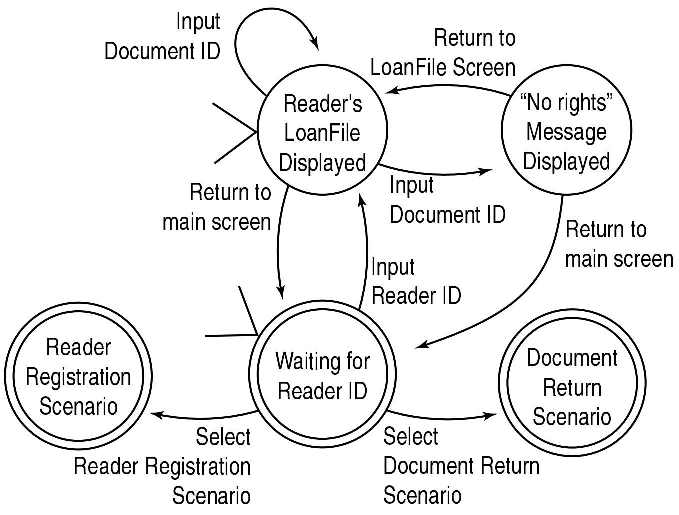
Input Document ID
Return to LoanFile Screen
Reader's LoanFile Displayed
"No rights" Message Displayed
Input Document ID
Return to main screen
Return to main screen
Input Reader ID
Reader Registration Scenario
Waiting for Reader ID
Document Return Scenario
Select Reader Registration Scenario
Select Document Return Scenario

Figure 3: Transition System for the Document Loan Scenario

- $Tc \subset Sc \times Hc \times Sc$ is the set of transitions. Transition $tc = (sc, hc, sc')$, means that while the scenario was in state sc, the user entered event hc, and one of the system's reactions is to put the scenario in state sc'.

A state-event automaton can be represented by a state-event diagram. Fig. 3 presents the state-event diagram for the Document Loan scenario.

4 Baseline Elicitation

In this step, scenario specifications are completed with data and other semantic information. At the end of the step, all informations required for scenario integration (i.e., the integration baseline) will be available, that is, entities and entity states, business rules, and correspondence between entity states and scenario states.

4.1 Data Modeling

Data modeling consists mostly in identifying (1) the "objects" (also referred to as entities) pertaining to a specific scenario, (2) the attributes describing these entities, (3) the domain of these attributes, and (4) the functional dependencies among these attributes. We rely on Two-Stage Entity-Relationship (TSER) [14] Methodology for these tasks. In the following section (Sect. 4.1.1), we briefly introduce TSER and its basic modeling tools. We then describe how TSER is used within the KLuB context to model data manipulated by the scenarios (Sect. 4.1.2).

4.1.1 The Two-Stage Entity-Relationship Model: TSER

The TSER [14] model was created to integrate functional analysis with database design. It entails two levels of models: a functional model, representing semantic content, and a structural model, forming a normalized data model. The purpose of the functional model is to describe data items and the relationships between these data items (i.e., functional dependencies) within specific contexts, while the purpose of the structural model is to provide a context-free representation of all the data items used within an information system. This context-free representation is a normalized data model which identifies entities, relationships between these entities, and attributes (i.e., data items) of these entities.

TSER provides rigorous algorithms which map from functional to structural models; these algorithms ensure that the resulting structures are at least in Boyce-Codd normal form (BCNF). TSER algorithms also integrate views, thus allowing systematic consolidation of any number of data models.

The Functional Model The functional model features semantic-level constructs for processes representation and for object-hierarchy. These constructs are used for system analysis and information requirements modeling. The constructs include the following:

- Subjects (fig. 4(a)) which represent functional units of information such as user views and application systems, and is analogous to frame or object. It contains, among other things, the attributes describing the subject (including their domains) and the functional dependencies describing relations between these attributes.
- Contexts (fig. 4(b)) which represent control knowledge and interactions among subjects.

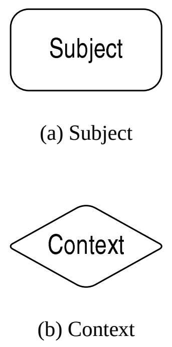

Figure 4: Functional Model Constructs

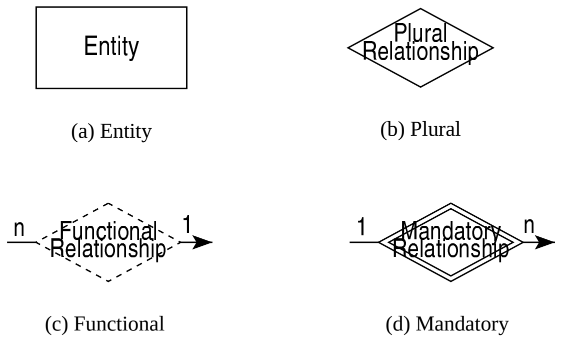

Figure 5: Structural Model Constructs

The Structural Model The structural model provides a normalized representation of data semantics from the functional model for logical database design. There are four basic constructs:

- Operational entities (or entities, for short; fig. 5(a)) which refer to constructs with a singular primary key.
- Plural relationships (fig. 5(b)) which refer to constructs with a composite primary key.
- Functional relationships (fig. 5(c)) representing referential integrity constraints between entities and/or plural relationships.
- Mandatory relationships (fig. 5(d)) representing existence dependency constraints between entities and/or plural relationships.

Normalized structures (i.e., the structural model) are obtained directly from the functional model, using three basic steps:

***Decomposition*:** creates a submodel for each subject in the subject hierarchy and analyzes its basic cardinality.

***Normalization*:** improves and simplifies the data structures within each submodel based on dependency theory. This step yields a model at least in BCNF.

***Consolidation*:** links and merges these submodels to produce a structural model corresponding to the functional model in the input. The Consolidation is performed recursively, starting from leaves in the subject decomposition tree, and creating a merged model for each intermediate node.

4.1.2 Using TSER in KLuB

In this research, we have opted for TSER (1) to model data within each scenario, creating one (possibly decomposed) subject per scenario, and (2) to obtain a normalized and integrated data model from these partial views. First, we construct a functional model for each scenario. This functional model describes the data items accessed by the scenario and the relationship between these data items. For complex scenarios, this modeling may involve multiple levels of TSER's functional model. For simple systems, this modeling may only involve the creation of a single subject.

For each of these functional models, we create a normalized data model (BCNF). This process breaks down TSER's subjects into normalized entities and relationships. For the purpose of this work, we will consider an entity any normalized construct having a primary key, either singular or composed (i.e., TSER's operational entities and plural relationships). By repeating this process to each scenario, we identify all the entities used in each scenario.

In the Document Loan scenario, we identified three "objects", namely, Document, LoanFile, and Reader. For each of these entities, we also identified attributes used in the scenario description, their respective domain, and the functional dependencies among them.

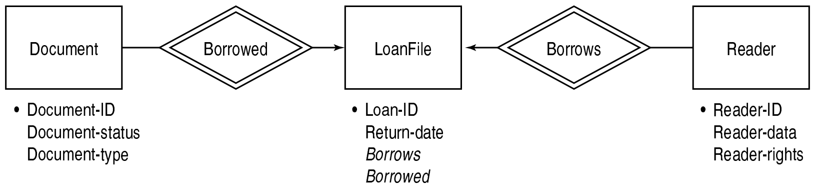

Figure 6: Library Integrated Structural Model

Document:	Document-ID:	$\{code1, \perp\}$
	Document-type:	$\{1, 2, \perp\}$
	Document-status:	$\{available, loaned, \perp\}$
Reader:	Reader-ID:	$\{code2, \perp\}$
	Reader-data:	$\{strings, \perp\}$
	Reader-rights:	$\{all\_documents, 1\_only, \perp\}$
LoanFile:	Loan-ID:	$\{code3, \perp\}$
	Return-date:	$\{dates, \perp\}$
	Borrows:	$\{code2, \perp\}$
	Borrowed:	$\{code1, \perp\}$

Figure 7: Attribute Domains for the Integrated Data Model

4.2 Data Models Integration

At this point, we produce an integrated data model, from the normalized data models obtained for each scenario (Sect. 4.1). The integrated data model is obtained by consolidating the structural models of all the scenarios. This task is automated using TSER's algorithms. The resulting model for the Library example is illustrated in fig. 6. Fig. 7 provides the integrated attribute domains. A special domain value noted "$\perp$" represents the value "not defined."

4.3 Semantics Elicitation

In this activity, more information on the scenario and on entities will be gathered, namely, business rules and in particular, pre- and postconditions.

4.3.1 Business Rules

In the system, business rules define how things should be done. They are constraints on data values, on functions, or on combinations of both, and are specified in predicate logic. Business rules are obtained from users, procedure manuals, the nature of the business tasks, and so on. In conjunction with the entities, they capture the application semantics, and will be used to construct the specification. Table 1 presents the rules from the Library system.

Table 1: Business Rules in the Library System

Rules	Description	Definition
r_1	If the Document entity does not exist, all its attributes do not exist	Document-ID = $\perp \rightarrow$ Document-status = $\perp \wedge$ Document-type = $\perp$
r_2	If the Document entity exists, some of its attributes must exist	Document-ID $\in code1 \rightarrow$ (Document-status = $available \vee$ Document-status = $loaned$) $\wedge$ (Document-type = 1 $\vee$ Document-type = 2)
r_3–r_6	These rules are similar to r_1 and r_2 for Reader and LoanFile entities	
r_7	If Reader is not registered, he (she) cannot borrow the document, and no LoanFile can exist for him (her)	Reader-ID = $\perp \rightarrow$ Loan-ID = $\perp \wedge$ Borrows = $\perp \wedge$ Borrowed = $\perp \wedge$ Document-status = $available$
r_8	If a document is in a LoanFile, the document's status must be "loaned"	Borrowed= Document-ID $\rightarrow$ Document-status = $loaned$
r_9	If Reader has a LoanFile, its reading rights must be compatible with the type of the loaned document	Borrows = Reader-ID $\wedge$ Borrowed = Document-ID $\rightarrow$ Reader-rights = $all\_documents \vee$ (Reader-rights = $1\_only \wedge$ Document-type =1)

4.3.2 Preconditions and Postconditions of Transitions

Preconditions are particular business rules which specify conditions under which a transition may be executed, while postconditions specify the effect of the transition on the entities. If a transition has no effect on entities, the precondition is also the postcondition of the transition. We will use $pre(tc)$ and $post(tc)$ to respectively represent the pre- and postcondition of a scenario transition tc.

4.3.3 Processing Associated with Transitions

Some transitions involve entity processing. For these transitions, the associated processing has to be specified. We note the set of such transition-associated processing $Pc = \{pc_1, ..., pc_n\}$, where pc_i is the processing associated to transition tc_i. Transition-associated processing is also specified by pre- and postconditions. The precondition specifies the values of the entities before the processing and may therefore be different from the transition precondition. In the case where no processing occurs, $pre(pc_i) = pre(tc_i)$ and $post(pc_i) = post(tc_i)$. The postcondition specifies the entity values resulting from the processing.

In the Document Loan scenario, only one transition (tc_5=(Reader's Loan File Displayed, Input Document ID, Reader's Loan File Displayed)) involves entity processing. The specification of the transition processing is made of:

$pre(pc_5)$**:** Document-status = $available \wedge$

Borrows $= \perp \wedge$ Borrowed $= \perp \wedge$ Loan-ID $= \perp$

$post(pc_5)$**:** Document-status = $loaned \wedge$

Borrows = Reader-Id $\wedge$ Borrowed = Document-ID $\wedge$ Loan-ID $\in code3$

4.4 Integration Baseline

The integration baseline is the set of results obtained in this step (Baseline Elicitation) and the preceding one (Scenario Acquisition). The system specification will be formally derived from that baseline which is composed of:

- entities as defined in the Data Models Integration activity (Sect. 4.2);
- FSAs of the scenarios (the Mc_i, Sect. 3.2);
- business rules (set R), as elicited in Section 4.3.1;
- transition pre- and postconditions, as identified in Section 4.3.2;
- specification of transition-associated processing as derived in Section 4.3.3.

5 Integration

The integration activities produce a complete system specification. The specification is formal and integrates data and behavior. All the activities are formal and can be automated.

5.1 Formal System Specification

For expressing the external specification of the system, a guarded sequential machine model is used. The system specification is defined as $SY = (E, Mc_i, Sy, Hy, Py, Ty, Iy, Fy, R)$, where

- E is the set of entities of the system, available from the baseline.
- Mc_i is the set of scenario automata for all scenarios $c_1, c_2, ..., c_n$, available from the baseline.
- Sy is the set of system states, and is derived from the entities, transitions pre- and postconditions, and from the scenario states, all available in the baseline. The algorithms are presented in Sections 5.2 and 5.3.
- Hy is the set of external events. $Hy = \bigcup Hc_i$, where Hc_i is the set of external events of scenario c_i, as specified in the scenario FSA part of the baseline (Mc_i).

- Py is the set of transition-associated processing. $Py = \bigcup Pc_i$, where Pc_i is the set of processes of scenario c_i, directly part of the baseline.
- $Ty \subset Sy \times Hy \times Py \times Sy$ is the set of transitions, and is derived from scenario transitions and scenario states, available in the baseline, from system states, entities and their states. The algorithm is described in Section 5.4.
- Iy is the set of initial states. Iy is a by-product of system state production; an initial system state is compatible with a scenario initial state.
- Fy is the set of final states. Fy is a by-product of system state production; a system final state is compatible with a scenario final state.
- R is the set of business rules, available from the baseline.

5.2 Data and Behavior Integration

In this section, scenario states and entities will be related by a formal model based on relations.

5.2.1 Concept of Entity State

Entity State Given the attributes $a_1, ..., a_n$ of entity e and the domain $Dom(e)$ (defined as $Dom(a_1) \times ... \times Dom(a_n)$) of entity e, a state sn of entity e is a subset of $Dom(e)$. Moreover, all states of e constitute a partition of $Dom(e)$.

Compatibility between entity state and scenario state Let $Sn(e) = \{sn_1, ..., sn_m\}$ be the set of states of entity e. For $i \in \{1 : m\}$, $sn_i = \{vn_{i1}, ..., vn_{in_i}\}$, where $vn_{ij} \in Dom(e)$. Let c be a scenario and sc be a state of scenario c.

Definition. A state sn of entity e is compatible with state sc of scenario c if, $\forall vn_i \in sn$, at least one of the following conditions is satisfied:

1. there exists a transition tc starting at sc, and vn_i satisfies $pre(tc)$;
2. there exists a transition tc' ending at sc, and vn_i satisfies $post(tc')$.
3. there exists an entity e', having a state sn' such that:
   - sn' is compatible with sc, as stated in Conditions 1 or 2 above;
   - there exists a business rule stating that e can be in state sn only when e' is in state sn'

4. the following conditions are all satisfied:
    - e is involved in no transition starting or ending at sc;
    - e is involved in no business rule as stated in Condition 3 above;
    - e is invisible in scenario state sc, and all its states are compatible with c.

Relation on $Dom(e)$. We first observe that the definition of compatibility given above can apply to a single element vn of $Dom(e)$. We define the relation Q on the values of $Dom(e)$ such that vn_iQvn_j iff $Sc_i = Sc_j$, where Sc_i and Sc_j are the sets of scenario states with which vn_i and vn_j are compatible, respectively.

It is easily shown that Q is an equivalence relation:

Q is reflexive: vn_iQvn_i means that $Sc_i = Sc_i$;

Q is symetric: if vn_iQvn_j, then $Sc_i = Sc_j$; therefore, $Sc_j = Sc_i$, which implies vn_jQvn_i;

Q is transitive: vn_iQvn_j implies $Sc_i = Sc_j$; vn_jQvn_k implies $Sc_j = Sc_k$; set equality (=) is transitive, therefore $Sc_i = Sc_k$, hence vn_iQvn_k.

Being an equivalence relation on $Dom(e)$, Q induces a partition of $Dom(e)$. The equivalence classes of the partition define the states sn_i of e.

Compatible set For each entity state sn, there is an associated set of scenario states $Comp(sn)$ with which all values in sn are compatible. $Comp(sn)$ is defined as the compatible set of sn.

5.2.2 Construction of Entity States at the Scenario Level

Composition of Relations on Partial Sets of Scenarios Let Sc_1, Sc_2, be subsets of a set Sc of scenario states. Let e be some entity and $Dom(e)$ the entity domain. The relation Q defined above induces on $Dom(e)$ one partition with respect to Sc_1 and one partition with respect to Sc_2:

- $\Pi(e) = \{sn_1, ..., sn_n\}$ is the partition induced on $Dom(e)$ by Q with respect to Sc_1 and $Comp(sn_1), ..., Comp(sn_n)$ are the corresponding compatible sets;
- $\Pi'(e) = \{sn'_1, ..., sn'_{n'}\}$ is the partition induced on $Dom(e)$ by Q with respect to Sc_2 and $Comp(sn'_1), ..., Comp(sn'_{n'})$ are the corresponding compatible sets.

Proposition 1 *The equivalence classes induced by Q with respect to $Sc_1 \cup Sc_2$ on $Dom(e)$ are $sn_i \cap sn'_j, \forall i, j$. Moreover, the corresponding compatible sets are $Comp(sn_i \cap sn'_j) = Comp(sn_i) \cup Comp(sn'_j)$.*

Proof

1. Compatibility of elements of $sn_i \cap sn'_j$:

   - $sn_i = \{vn_1, vn_2, ..., vn_m\}(vn_k \in Dom(e))$;
   - $sn'_j = \{vn'_1, vn'_2, ..., vn'_{m'}\}(vn'_l \in Dom(e))$;

   It is obvious that elements common to sn_i and sn'_j ($sn_i \cap sn'_j$) are compatible with $Comp(sn_i)$ and with $Comp(sn'_j)$. Moreover, because of the definition of Q, $Comp(sn_i)$ contains the only elements of Sc_1 with which $sn_i \cap sn'_j$ are compatible, and $Comp(sn'_j)$ contains the only elements of Sc_2 with which $sn_i \cap sn'_j$ are compatible. Therefore, $Comp(sn_i) \cup Comp(sn'_j)$ contains the only elements of Sc with which $sn_i \cap sn'_j$ are compatible. The partition of Sc including $sn_i \cap sn'_j$ may contain more elements of $Dom(e)$ but its compatible set is exactly $Comp(sn_i) \cup Comp(sn'_j)$.

2. Compatible set of the partition with respect to $Sc_1 \cup Sc_2$:

   Let us consider the partition of $Dom(e)$ based on $Comp(sn_i) \cup Comp(sn'_j)$. Let $sn = \{vn_1, vn_2, ..., vn_m\}, vn_k \in Dom(e)$, be an element of the partition and let $Comp(sn)$ be its compatible set.

   - The element vn_k belongs to one and only one element of the partition with respec t to Sc_1. Let sn_i be that element, with $Comp(sn_i)$ the corresponding compatible set.
   - The element vn_k belongs to one and only one element of the partition with respec t to Sc_2. Let sn'_j be that element, with $Comp(sn'_j)$ the corresponding compatible set.

   Therefore, vn_k belongs to $sn_i \cap sn'_j$ and is compatible with $Comp(sn_i) \cup Comp(sn'_j)$. Because of the definition of relation Q, it is easy to show that $Comp(sn_i) \cup Comp(sn'_j)$ are the only elements of $Sc_1 \cup Sc_2$ with which vn_k is compatible. Therefore $Comp(sn_i) \cup Comp(sn'_j) = Comp(sn)$. Because this is true for any $vn_k\imath nsn$, all elements of sn are common to sn_i and sn'_j.

Construction of Entity States Entity States are constructed by applying the composition of relations property defined above, one scenario state at a time. The process involves two steps: (1) for each entity, defining the values compatible with each scenario state; (2) for each entity, calculating the entity states.

Table 2: Entity States for the Library System

Entity	State	Compatiblity set	Meaning
Document (e_1)	sn_{11}	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}\}$	Document-status = $available$ $\wedge$ Document-type = 1
Document (e_1)	sn_{12}	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}, sc_{15}\}$	Document-status = $available$ $\wedge$ Document-type = 2
Document (e_1)	sn_{13}	$\{sc_{11}, sc_{12}, sc_{14}\}$	Document-status = $loaned$
Document (e_1)	sn_{14}	$\emptyset$	Document-ID = $\perp$
Reader (e_2)	sn_{21}	$\{sc_{11}, sc_{12}, sc_{13}\}$	Reader-ID = $\perp$
Reader (e_2)	sn_{22}	$\{sc_{11}, sc_{12}, sc_{14}\}$	Reader-ID $\in code2$ $\wedge$ Reader-rights = $all\_documents$ $\wedge$ Reader-data $\in strings$
Reader (e_2)	sn_{23}	$\{sc_{11}, sc_{12}, sc_{14}, sc_{15}\}$	Reader-ID $\in code2$ $\wedge$ Reader-rights = 1_only $\wedge$ Reader-data $\in strings$
Loan File (e_3)	sn_{31}	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}, sc_{15}\}$	LoanFile-ID = $\perp$
Loan File (e_3)	sn_{32}	$\{sc_{11}, sc_{12}, sc_{14}\}$	LoanFile-ID $\in code3$ $\wedge$ Borrows $\in code2$ $\wedge$ Borrowed $\in code1$

The construction of entity states proceeds using the *Entity State Generation Algorithm* (see Appendix). At a scenario state, the basic partition of an entity involves two sets: values compatible with the scenario state, and values not compatible with the state (this set may be empty). The basic partitions can therefore be constructed directly when finding the entity values compatible with a scenario state, using the definition of compatibility given above. These basic partitions are then merged, one scenario state at a time, using the results from Proposition 1. The *Entity State Generation Algorithm* constructs the basic partitions, based on the business rules identified, and merges these partitions in order to obtain the entity states.

The results are all states of the entity and for each entity state, its compatible set. Table 2 shows the results for the Library system.

5.3 System States Generation

This activity generates the states of the system specification, based on the entity states and the scenario states.

5.3.1 System State Definition

A system state is characterized by the values of the entities composing the system. Given $E = \{e_1, ..., e_n\}$, a set of system entities, and $Sn(e_i)$, the set of entity states for entity e_i, we have the set of *potential system states* W, defined as $Sn(e_1) \times ... \times Sn(e_n)$.

Some of the potential system states are invalid, based on the business rules (Sect. 4.3). A *purified potential system state*, noted wp, is a potential system state which agrees with all the business rules identified in all the scenarios. Given $R =$

$\{r_1, ..., r_m\}$, a set of business rules, we have the set of purified potential system states Wp, which is defined as:

$$\{w = (sn_1, ..., sn_n) \in W \mid r_1 \wedge ... \wedge r_m\}$$

System State A system state sy is a subset of Wp. Moreover, all system states constitute a partition of Wp.

Compatibility between system state and scenario state A system state $sy = \{wp_1, ..., wp_m\}$ is compatible with scenario state sc if, for every purified potential system state $wp_i = (sn_{i1}, ..., sn_{in})$, every entity state sn_{ij} is compatible with sc.

Relation on Wp. We observe that the definition of compatibility given above can apply to element wp of sy. We define the relation Q' such that $wp_i Q' wp_j$ iff $Sc_i = Sc_j$, where Sc_i and Sc_j are the sets of scenario states with which wp_i and wp_j are compatible, respectively.

Again, it is easily shown that Q' is an equivalence relation. Therefore, it induces a partition on Wp. The equivalence classes of the partition define the system states Sy.

Compatible set For each system state sy_i, there is an associated set of scenario states $Comp(sy_i)$ with which all values in sy_i are compatible. $Comp(sy_i)$ is defined as the compatible set of sy_i. There is no constraint on the content of $Comp(sy_i)$. It could be empty. Therefore, one equivalence class can correspond to those values of Wp which are compatible with no scenario state of Sc, the set of all scenarios. These states represent invalid states in the system.

5.3.2 Integration Algorithm

Integrating entity states to produce system states is in fact equivalent to a cartesian product of $\| E \|$ sets, each set having on average $avg = \sum_{i=1}^{\|E\|} \frac{Sn(e)}{\|E\|}$ elements. Furthermore, each of the resulting tuples must be checked against each business rule, to ensure validity of the entity states combination. Therefore, calculating systems states may require $O(avg^{\|E\|} \cdot \| R \|)$.

Obviously, we want to reduce this number of steps as much as possible. The strategy that we have developed uses the approach used in relational database systems (RDBS) to optimize queries [15]. In RDBS, the "query plan" usually consists of a tree where the leaves are the relations to join to obtain the final query result, and the nodes are join operations to perform.

The integration involves two tasks. The first task, the *Cartesian Product Sorting Algorithm*, generates the "query plan", while the second task, the *System State Generation Algorithm*, generates the system states by applying that "query plan".

Cartesian Product Sorting Algorithm This algorithm (see the Appendix) constructs the cartesian product evaluation tree. Each leaf is the set of states for one entity. Each node is a cartesian product between two subtrees, constrained by some business rules. The algorithm performs in $O(\| E \|^4)$.

We iterate until we have determined the order of all the cartesian products. At each iteration, we determine the best product to perform, based on the expected number o of tuples in the partial result. Assuming that π and π' are partial results, $o(\pi \cup \pi')$ is the number of tuples in the cartesian product of π and π', given by $o(\pi) \cdot o(\pi') - \sum_{e \in \pi \cup \pi'} \frac{avg(e)}{2} \cdot rules(\pi \cup \pi')$, where $avg(e)$ is the average number of tuples for entity e and $rules(\pi \cup \pi')$ is the set of rules involving only the entities in π and π'.

In the Library system, based on the results from the algorithm, we would first integrate entities Document and LoanFile. Then, we would integrate this partial result with the Reader entity.

System States Generation Algorithm This algorithm (see the Appendix) generates the purified potential system states and removes invalid states as early as possible. It incrementally creates these states by following the cartesian product order determined by the *Cartesian Product Sorting Algorithm*. At each increment, we merge two previously obtained partial products. At the end, we create system states by partitionning the purified potential system states on the set of compatible scenario states.

In the Library system, when only considering the Document Loan scenario, we obtain the following system states (fig. 8):

- sy_1: LoanFile does not exist, Reader does not exist, Document is *available*;
- sy_2: Reader exists and either LoanFile does not exist and Document is *available* and of type 1, or Document is loaned;
- sy_3: Reader exists, LoanFile does not exist and Document is *available* and of type 2.

5.4 Generating External System Transitions

5.4.1 Definition and Properties

A system transition is a tuple (sy, hy, py, sy'). Each such system transition corresponds to a scenario transition (sc, hc, sc'), scenario states sc and sc' being in the compatible set of sy and sy', respectively, provided that (1) sy does not violate the precondition of the scenario transition, (2) sy' does not violate the postcondition of the transition, and (3) that any entity not involved in the scenario transition remains unchanged.

Given a scenario transition and a pair of system states, this can be verified automatically, because the required information is available either in the baseline (pre- and postconditions) or as result of the system state generation step (entity states corresponding to a system state).

In fig. 8, the part of the system specification produced by the Document Loan Scenario is shown. It is to be noted that a scenario transition may create several system transitions. It should also be noted that some processing is missing from this

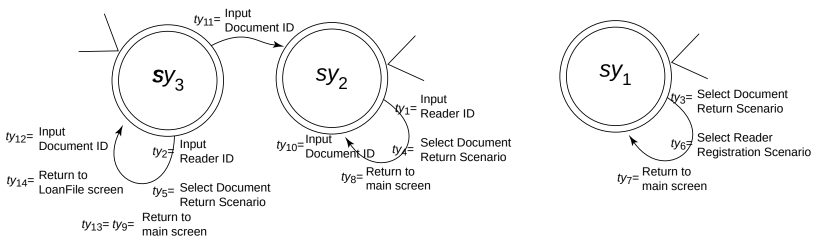
ty_{11}= Input Document ID
sy_3
sy_2
sy_1
ty_1= Input Reader ID
ty_3= Select Document Return Scenario
ty_{12}= Input Document ID
ty_2= Input Reader ID
ty_{10}= Input Document ID
ty_4= Select Document Return Scenario
ty_6= Select Reader Registration Scenario
ty_{14}= Return to LoanFile screen
ty_5= Select Document Return Scenario
ty_8= Return to main screen
ty_7= Return to main screen
ty_{13}= ty_9= Return to main screen

Figure 8: Final System Automaton

system:

- No transition from sy_1 to the other states. A reader must register before he/she can borrow book. Clearly, the registration scenario is missing.
- No transition from sy_2 to sy_3. This follows from the absence of the Return scenario.

6 Discussion/Conclusion

A major objective of this work was to integrate data and behavior into a single, formal, specification. This implied elicitation and formal specification of the data involved in the system. By using the TSER approach, we were able to specify formally the data involved in each scenario. Data integration at the system level was also performed with TSER, resulting in a formal, i.e. third normal form, data model. Complete semantic integration of data and behavior was formally defined by the concepts of entity states and of compatibility between entity/system state and scenario state. The introduction of business rules expanded the semantic part of the requirements. The integration step, completely formal, yields a formal specification based on an extended finite state machine.

The potential combinatorial explosion of the integration algorithm is controlled by well-tried methods drawn from database processing, and semantic information provided by the business rules. The concrete library example used in this work was larger than most presented in the literature, and the combinatorial explosion was well controlled. The risk exists nonetheless, but there are now systems which can handle finite state machines with thousand of states. One could also question the entity state concept with data having continuous attributes instead of discrete ones like in the library example. As a matter of fact, the formal definitions of entity states make no assumption on the domains of the attributes, and the entity states construction algorithm can easily be adjusted to continuous values.

Our approach has some limitations. It handles only one instance of each entity and one instance of each scenario. Also, the approach implies that scenarios can only be executed sequentially. Work currently under way addresses these limitations, by considering several instances of entities, the possibility of multiple instances of a scenario running concurrently, the possibity of scenarios running concurrently.

Finally, because the approach is formal, it has a strong potential for verification and validation. The formal approach, manual or automatic, is a fertile ground for performing verifications along the requirements elicitation process and not at the end, on the result, as is usually the case. Work is underway to exploit that formal verification potential.

References

[1] A. Dardenne, A. van Lamsweerde & S. Fickas, Goal-directed requirements acquisition, *Science of Computer Programming*, *20* (1-2), 1993, 3-50.

[2] H. Holbrook III, A scenario-based methodology for conducting requirements elicitation, *ACM SIGSOFT Software Engineering Notes*, *15* (1), 1982, 95-104.

[3] UML notation guide, version 1.1, 1997.

[4] I. Jacobson, M. Christerson, P. Jonsson & G. Overgaard, *Object-Oriented Software Engineering - A Use Case Driven Approach* (Addison-Wesley, revised edition, 1994).

[5] J. Desharnais, M. Frappier, R. Khédri & A. Mili, Integration of sequential scenarios, *IEEE Transactions on Software Engineering*, *24* (9), 1998, 695-708.

[6] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyohima & C. Chen, Formal approach to scenario analysis, *IEEE Software*, *11*, 1994, 33-41.

[7] I. Kawashita, Spécification formelle des systèmes d'information par la technique des scénarios, masters diss., Université de Montréal, Montréal, Québec, Canada, 1997.

[8] S. Somé, R. Dssouli & J. Vaucher, From scenarios to timed automata: Building specifications from users requirements, *2nd Asia Pacific Software Engineering Conference APSEC 95*, 1995.

[9] I. Khriss, M. Elkoutbi & R. Keller, Automating the synthesis of statechart diagrams from multiple collaboration diagrams, *Proc. International Workshop on the Unified Modeling Language "UML"'98 : Beyond the Notation*, 1998, 115-126bis.

[10] M. Glinz, An integrated formal method of scenarios based on statecharts, *Lecture Notes in Computer Science - Proceedings of the European Conference in Software Engineering 1995*, (989), 1995, 254-271.

[11] F. Lustman, A formal approach to scenario integration, *Annals of Software Engineerig*, *3*, 1997, 255-272.

[12] K.S. Rubin & A. Goldberg, Object behavior analysis, *Communications of the ACM*, *35* (9), 1992, 48-62.

[13] P. Shoval, ADISSA: Architectural design of information system based on structured analysis, *Information Systems*, *13*, 1998, 193-210.

[14] C. Hsu, Y. Tao, M. Bouziane & G. Babin, Paradigm translation in manufacturing information using a meta-model: The TSER approach, *Ingénierie des systèmes d'information*, *1* (3), 1993, 325-352.

[15] C.J. Date, *An Introduction to Database Systems*, volume 1 of *The Systems Programming Series*, (Addison-Wesley, fifth edition, 1990).

A Appendix

Algorithm 1 *Entity State Generation Algorithm.*

```
    Let $C = \{c_1, c_2, ..., c_n\}$ be the set of all scenarios of the system;
    Let $Sc_i = \{sc_{i1}, ..., sc_{im}\}$ be the set of states of scenario $c_i$;
    Let $Sn(sc)$ be the set of entity states based on the compatibility of scenario state sc;
```

Let Sn_{temp} be a temporary set of entity states;
Let Sn be the final set of all entity states;
Let sn, sn' be entity states;
Let $Comp(sn)$ be the set of scenario states compatible with entity state sn.

A- We first generate basic partitions for each scenario state of each scenario.

For each $sc_{ij}, i \in \{1, ..., n\}, j \in \{i, ..., m\}$

We create a partition $Sn(sc_{ij})$ of $Dom(e)$, based on the compatibility of sc_{ij}. $Sn(sc_{ij})$ will be made of two elements, sn containing those elements of $Dom(e)$ compatible with sc_{ij} and sn' containing those elements of $Dom(e)$ which are not compatible with sc_{ij}.

```
    $sn \leftarrow$ subset of $Dom(e)$ compatible with $sc_{ij}$
    $sn' \leftarrow$ subset of $Dom(e)$ not compatible with $sc_{ij}$
    If $sn' \neq \emptyset$
        $Sn(sc_{ij}) \leftarrow \{sn, sn'\}$
        $Comp(sn) \leftarrow \{sc_{ij}\}$
        $Comp(sn') \leftarrow \emptyset$
    Else
        $Sn(sc_{ij}) \leftarrow \{sn\}$
        $Comp(sn) \leftarrow \{sc_{ij}\}$
    End-If
End-For
```

B- Starting with one scenario state, the partitions are composed as seen above, by adding each time one scenario state. The result is:

- all states of the entity;
- for each entity state, its compatible set.

```
$Sn \leftarrow \{Dom(e)\}$
$Sn_{temp} \leftarrow \emptyset$
For each $sc_{ij}, i \in \{1, ..., n\}, j \in \{i, ..., m\}$
    For each $sn \in Sn$
        For each $sn' \in Sn(sc_{ij})$
            If $sn \cap sn' \neq \emptyset$
                $Sn_{temp} \leftarrow Sn_{temp} \cup \{sn \cap sn'\}$
                $Comp(sn \cap sn') \leftarrow Comp(sn) \cup Comp(sn')$
            End-If
        End-For
    End-For
    $Sn \leftarrow Sn_{temp}$
    $Sn_{temp} \leftarrow \emptyset$
End-For
```

Algorithm 2 *Carthesian Product Sorting Algorithm.*

Let $E = \{e_1, ..., e_n\}$ be the set of entities;
Let $Sn(e)$ be the set of states of entity e;
Let $avg(e, j)$ be the average number of occurences of a state of entity e after the j^{th} carthesian product;
Let $R = \{r_1, ..., r_m\}$ be the set of business rules;
Let $\Pi = \{\pi_1, ..., \pi_l\}$ be a partition on E;
Let Π' be a partition on E;
Let $rule(\{e_1, ..., e_n\})$ be the number of rules involving entities $e_1, ..., e_n$; this function is precalculated;
Let $o(\{e_1, ..., e_k\})$ be the number of operations needed to perform the carthesian product of entities $e_1, ..., e_k$;
Let Max be the largest possible number of operations; $Max \triangleq \| Sn(e_1) \| \cdot ... \cdot \| Sn(e_n) \|$
Let Min be the smallest number of operations to perform during an iteration;
Let $Ord_1(i)$ be the first set of entities being merged at the i^{th} carthes ian product;
Let $Ord_2(i)$ be the second set of entities being merged at the i^{th} carthe sian product.

A- We initialize Max, Π, avg, and o.

```
$Max \leftarrow 1$
$\Pi \leftarrow \emptyset$
**For each** $i \in \{1, ..., n\}$
    $\pi_i \leftarrow \{e_i\}$
    $\Pi \leftarrow \Pi \cup \{\pi_i\}$
    $avg(e_i, 0) \leftarrow 1$
    $o(\pi_i) \leftarrow \| Sn(e_i) \|$
    $Max \leftarrow Max \cdot o(\pi_i)$
**End-For**
```

We iterate until we have determined the order of all the carthesian products. At each iteration, we determine the best product to perform. The variable Ord preserves that information for the *System State Generation Algorithm*.

```
$i \leftarrow 1$
**While** $\| \Pi \| \neq 1$
```

B- We calculate the cost (o) for each partition to determine the best initial merge. We also determine the carthesian product that will yield the smallest number of operations. The general equation for o is:

$$o(\pi_j \cup \pi_k) = o(\pi_j) \cdot o(\pi_k) - \sum_{e \in \pi_j \cup \pi_k} \frac{avg(e, i-1)}{2} \cdot rules(\pi_j \cup \pi_k)$$

```
    $Min \leftarrow Max$
    **For each** $j \in \{1, ..., \| \Pi \| - 1\}$
        $Avg_1 \leftarrow 0$
        **For each** $e \in \pi_j$
            $Avg_1 \leftarrow Avg_1 + avg(e, i-1)/2$
        **End-For**

        **For each** $k \in \{j, ..., \| \Pi \|\}$
            $Avg_2 \leftarrow Avg_1$
            **For each** $e \in \pi_k$
                $Avg_2 \leftarrow Avg_2 + avg(e, i-1)/2$
            **End-For**

            $o(\pi_j \cup \pi_k) \leftarrow o(\pi_j) \cdot o(\pi_j) - Avg_2 \cdot rules(\pi_j \cup \pi_k)$

            **If** $Min > o(\pi_j \cup \pi_k)$
                $Min \leftarrow o(\pi_j \cup \pi_k)$
                $Ord_1(i) \leftarrow \pi_j$
                $Ord_2(i) \leftarrow \pi_k$
            **End-If**
        **End-For**
    **End-For**
```

C- We calculate the average number of occurences of an entity after this iteration (avg). When we merge $\pi (= Ord_1(i))$ and $\pi' (= Ord_1(i))$, for $e \in \pi \cup \pi'$, we have:

$$avg(e, i) = \frac{o(\pi \cup \pi')}{\| Sn(e) \|}$$

```
    **For each** $e \in Ord_1(i) \cup Ord_2(i)$
        $avg(e, i) \leftarrow \frac{o(Ord_1(i) \cup Ord_2(i))}{\| Sn(e) \|}$
    **End-For**
    **For each** $e \in E \setminus (Ord_1(i) \cup Ord_2(i))$
```

```
            $avg(e,i) \leftarrow avg(e,i-1)$
        End-For

        D- We merge the partitions corresponding to the previous carthesian product.

        $k \leftarrow 1$
        $\Pi' \leftarrow 1$
        For each $j \in \{1,...,\|\Pi\|\}$
            If $\pi_j = Ord_1(i)$
                $\pi'_k \leftarrow Ord_1(i) \cup Ord_2(i)$
                $\Pi' \leftarrow \Pi' \cup \{\pi'_k\}$
                $k \leftarrow k+1$
            Else-If $\pi_j \neq Ord_2(i)$
                $\pi'_k \leftarrow \pi_j$
                $\Pi' \leftarrow \Pi' \cup \{\pi'_k\}$
                $k \leftarrow k+1$
            End-If
        End-For
        $\Pi \leftarrow \Pi'$
        $i \leftarrow i+1$
    End-While
    $Ord_1(i) \leftarrow E$
```

Algorithm 3 *System States Generation Algorithm.*

Let $E = \{e_1,...,e_n\}$ be the set of entities;
Let $Sn(e)$ be the set of states of entity e;
Let $R = \{r_1,...,r_m\}$ be the set of business rules;
Let $rule(\{e_1,...,e_n\})$ be the number of rules involving entities $e_1,...,e_n$; this function is precalculated;
Let $Ord_1(i)$ be the first set of entities being merged at the i^{th} carthes ian product resulting from the *Carthesian Product Sorting Algorithm*;
Let $Ord_2(i)$ be the second set of entities being merged at the i^{th} carthe sian product resulting from the *Carthesian Product Sorting Algorithm*;
Let $W(\{e_1,...,e_n\})$ be the purified potential system states in the carthesian produc t of entities $e_1,...,e_n$;
Let Sc be the set of scenario states;
Let $Comp(sn)$ be the set of scenario states compatible with entity state sn;
Let $Comp(w)$ be the set of scenario states compatible with purified potential system state w;
Let $Sy = \{sy_1,...,sy_l\}$ be the set of system states; each system state is a set of purified potential system states;
Let $Comp(sy)$ be the set of scenario states compatible with system state sy.

```
A- We initialize $Next'$, $CompTuple$, $W$, $next$, and $previous$.

For each $e \in E$
    $W(\{e\}) \leftarrow Sn(e)$
End-For

B- We perform the carthesian product.

$i \leftarrow 1$
While $Ord_1(i) \neq E$
    For each $w \in Ord_1(i)$
        For each $w' \in Ord_2(i)$
            $OK \leftarrow \mathsf{True}$
            For each $r \in rules(Ord_1(i) \cup Ord_2(i))$
                $OK \leftarrow OK \wedge r(w,w')$
                If $\not OK$
                    Exit For each loop
                End-If
            End-For
            If $OK \wedge (Comp(w) \cap Comp(w') \neq \emptyset)$
```

```
                $W(Ord_1(i) \cup Ord_2(i))) \leftarrow W(Ord_1(i) \cup Ord_2(i))) \cup w \bowtie w'$
                $Comp(w \bowtie w') \leftarrow Comp(w) \cap Comp(w')$
            **End-If**
        **End-For**
    **End-For**
    $i \leftarrow i + 1$
**End-While**
```

C- We create system states by partitionning the purified potential system states o n the set of compatible scenario states.

```
$Sy \leftarrow \emptyset$
**For each** $w \in W(E)$
    $found \leftarrow \mathsf{False}$
    **For each** $sy \in Sy$
        **If** $Comp(w) = Comp(sy)$
            $sy \leftarrow sy \cup \{w\}$
            $found \leftarrow \mathsf{True}$
            Exit For each loop
        **End-If**
    **End-For**
    **If** $\neg found$
        $sy \leftarrow \{w\}$
        $Comp(sy) \leftarrow Comp(w)$
        $Sy \leftarrow Sy \cup \{sy\}$
    **End-If**
**End-For**
```

Biographies

Gilbert Babin received the B.Sc and M.Sc. from Université de Montréal (Canada), and the Ph.D. from Rensselaer Polytechnic Institute (USA). From 1993 to 2000, he has been a faculty member at Université Laval (Canada). He his currently Associate Professor at the École des Hautes Études Commerciales de Montréal (Canada). He has been working on the integration of heterogeneous, distributed databases using reactive agents and a central knowledge repository. Some of his work has been published in IEEE Transactions. His current interests include the Web Operating System (WOS^{TM}), trust management and e-commerce.

François Lustman received a BSc in Electrical Engineering and a PhD in Applied Mathematics from University of Grenoble, France. He is currently professor at the Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada. Before joigning Universit'e de Montréal, he spent fifteen years working in private and public organizations. He joigned the University in 1981 and chaired the Department from 1985 to 1989. He is a member of the Gelo software engineering group and his present research interests are in software quality and evolvability, and in the use of formal methods for specifying and designing information systems. François Lustman has published one book on software project management, and papers in compiler construction, medical databases, project management, information systems, software quality, and formal methods.