François Lustman and Gilbert Babin. Formal Data and Behavior Requirements Engineering: A Scenario-based Approach. Research report no. DIUL-RR-9901. Université Laval. Ste-Foy, Québec, Canada. January 1999.

Formal Data and Behavior Requirements Engineering : A Scenario-based Approach

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

Gilbert Babin
Département d'informatique
Université Laval
Québec, Canada G1K 7P4
babin@@ift.ulaval.ca

January 20, 1999

1 Introduction

In requirements engineering, the use of formal methods is often limited to the result of the elicitation process, i.e., the system specification. The main objective of the Forspec project is to use formal methods not only at the specification level but also along the requirements elicitation process. Other objectives are to integrate data, process, and user interface into a single formal specification. The work reported below, deals with integrating data and behavior into a single formal specification. The requirements elicitation approach selected is scenario-based.

Scenarios have been recognized as an effective technique for eliciting requirements in general [2, 3, 7], for investigating behavior, in particular in the object-oriented approach [10, 15, 17]. Scenario description, first informal, has lately been overtaken by more formal graphical approaches [10, 15, 17]. Tools like finite state automata have been used in many variations [8, 11, 19, 20]. Scenarios have also been represented as relations [5].

What does a scenario describe? Usually behavior [8, 10, 12, 15, 17], sometimes data and behavior [5, 11, 19, 20]. A common feature of the different definitions of a scenario 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, 6, 14], in others it does not [11, 19, 20]. The integration technique is manual [5, 6], or algorithmic [11, 19, 20], or human-assisted [12, 14].

Scenario description raises no problem anymore, as long as the description is limited to behavior. If the description is supposed to involve data as well, definition and integration problems arise. How is data elicited and described? How are data and behavior integrated? At the system level, several questions remain unanswered or uncompletely answered. Which formal tool should be used to describe behavior and data in an integrated way? How are data integrated? What about system semantic not expressed in scenarios? If an algorithm is used to integrate scenarios, how to avoid or control potential combinatorial explosions?

While this work does not pretend to solve all the above-mentioned problems, it addresses several of them in the specific framework, of information systems. The overall objective of the KluB approach presented below, is to use formal techniques as much as possible to describe scenarios and to integrate them. Specific objectives are :

- formal modeling of data and behavior at the scenario level,
- introduction of system semantic through business rules,
- semantic integration of data and behavior at the scenario and system levels,
- algorithmic integration of scenarios,
- use of well-known techniques for controlling the potential combinatorial explosion.

The approach is limited to sequential scenarios and to one instance of each data object, one instance of each scenario.

An overview of the KluB approach is presented in Section 2, as well as an introduction to TSER, the formal technique used to elicit and integrate data. The scenario definition and formal description used in this work, are provided in Section 3. Section 4, Scenario Analysis, deals with semantics. Data modeling is described as is the concept and formalism of business rules. Section 5 presents the two-steps integration process. The formal tool used to specify a system is introduced. Then, in the first integration step, a formal method for integrating data and behavior at the scenario level is described. The second step performs the overall data-based, system integration. Finally, in the Conclusion (Sect. 7), potential benefits and problems are discussed and possible extensions are presented.

2 Method Overview

2.1 Formal Framework

Basic formal tools are used for modeling the various parts of scenarios and of the system specification. Integrating data and behavior in a single specification is a major part of this work. For data modeling, TSER (Two-stage entity-relationship), an E/R-based model and process, well suited for data consolidation, will be used. For behavior modeling, variations of finite state machines (FSA) are required. The behavior part of a scenario will be modeled with state-event diagrams, while the system specification will be expressed by a guarded sequential machine. Finally first-order predicate logic is all that is needed to express pre and postconditions and business rules.

2.2 Requirements Engineering

A major objective of the Forspec project is to integrate data and behavior in a single specification or, to state in another way, to link the syntactical model of a scenario (the FSA) with a semantic model. The KLuB sub-project is concerned with integrating scenarios. In this work, an extension of [11], the integration is performed not on the scenario FSAs but on the scenarios' semantics composed of entities and business rules. The glue for tying syntax and semantics together is provided by relating entity states to scenario and system states.

The overall requirements engineering process, outlined in Figure 1, comprises three steps :

1. Scenario acquisition
2. Scenario analysis
3. Scenario integration

The KluB sub-project is about scenario analysis and integration, and therefore covers steps 2 and 3.

Scenario acquisition, performed on a scenario by scenario basis, is an informal step, involving analysts and users. Because several systematic approaches for scenario elicitation are available (see [8] for example), this step is assumed to have been performed. For the example used in this work and described below (see Section 2.3), extensive interviews were held with users and managers, and recorded on tape.

Scenario analysis is performed scenario by scenario. It involves four steps, behavior modeling, data modeling, data model integration, and semantics elicitation. In the scenario world, a formal process for converting the informal requirements into a formal specification is available ([8]). As a consequence, this work does not deal with behavior modeling of a scenario. It is assumed that for each scenario, an informal description and the corresponding FSA, are available. Data visible in the scenario is elicited and modeled using the first stage of TSER. The step, partly formal, results in a formal data model, and is described in Section 4.1. In the data model integration activity, presented in Section 4.2, the data models of all scenarios are integrated, using the second stage of TSER. The result is an entity-relationship model of all system data, in third normal form. The semantics elicitation step consists in extracting and formalizing business rules from all relevant sources (informal description, users, procedure manuals), and in establishing the pre- and postconditions of scenario states. The step, described in Section 4.3, is informal but all results are formal.

Scenario integration produces the system specification. This step is formal, and consists of three activities. Integration of data and behavior is achieved by defining a relationship between scenario states and entity values. The process is formal as is the result (see Sect. 5.2-5.4). In the system state generation activity, described in Section 5.5,

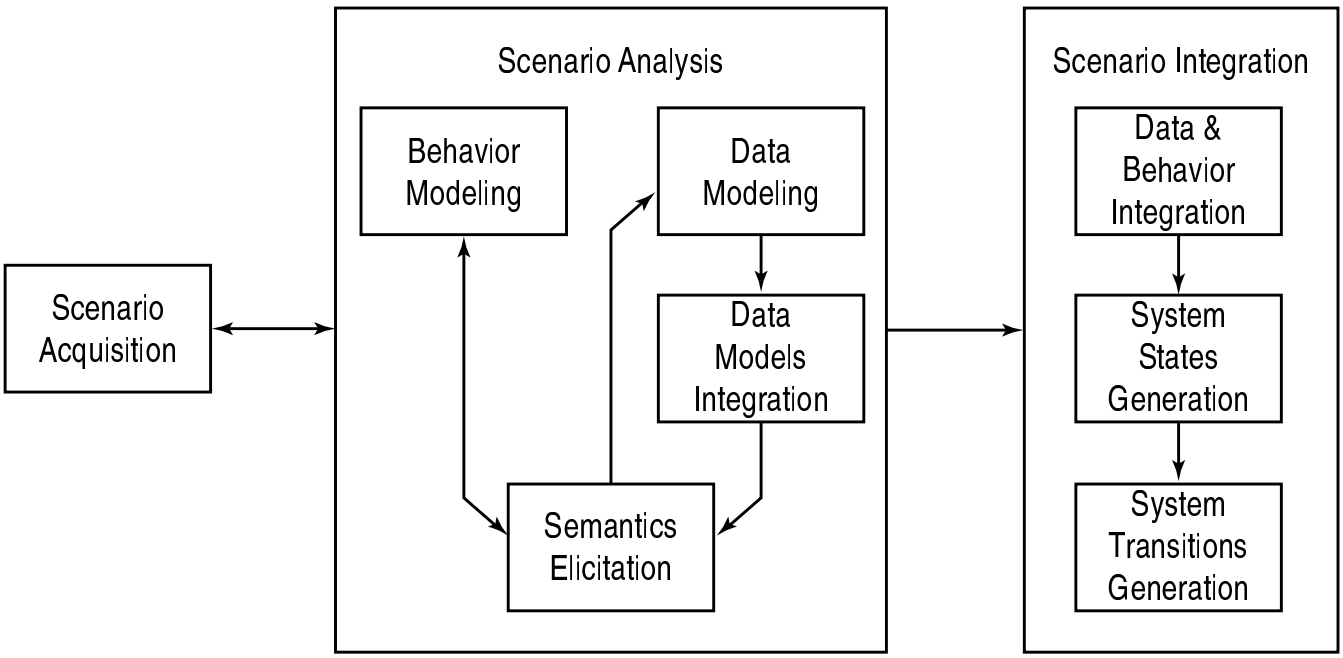


Figure 1: Requirements Engineering Process

entity values are incrementally combined to construct potential system states, and pruned using business rules and scenario compatibility rules. The result is the set of system states. In the last activity, described in Section 5.6, system transitions are generated from the scenario transitions, by considering the compatibility between transition pre- and postconditions and system states.

2.2.1 The Two-Stage Entity-Relationship Model : TSER

The Two-Stage Entity-Relationship (TSER) [9] 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. Furthermore, there are rigorous TSER algorithms which map from functional to structural models; these algorithms ensure that the resulting structures are at least in third normal form (3NF). 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. 2(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. 2(b)) which represent control knowledge and interactions among subjects.

In TSER, a functional dependency between two sets of attributes $A = \{a_1, ..., a_n\}$ and $A' = \{a'_1, ..., a'_m\}$, noted $A \rightarrow A'$, is defined as follows :

$$A \rightarrow A' \Leftrightarrow \forall (v_1, ..., v_n) \in Dom(A), \exists! (v'_1, ..., v'_m) \in Dom_{\perp}(A')$$
$$\text{such that } (v'_1, ..., v'_m) \text{ may be determined by } (v_1, ..., v_n),$$

where

$$Dom(A) \triangleq Dom(a_1) \times ... \times Dom(a_n)$$

(a) Subject (b) Context

Figure 2: Functional Model Constructs

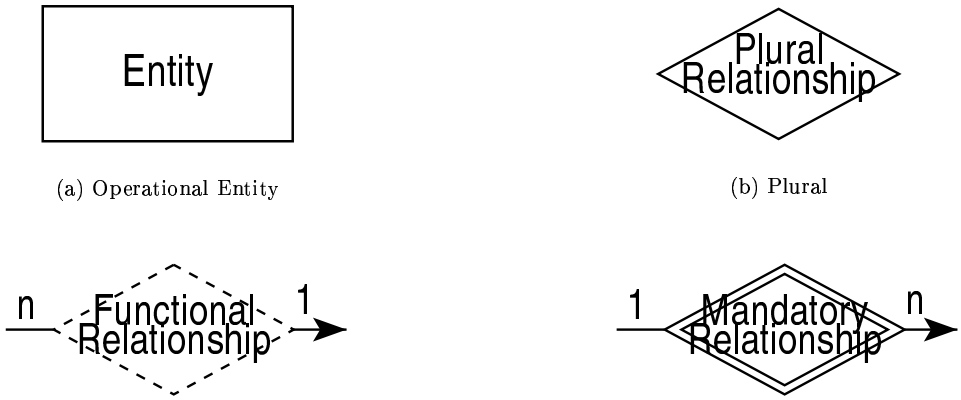
and

$$Dom_{\perp}(A') \triangleq (Dom(a'_1) \cup \{\perp\}) \times ... \times (Dom(a'_m) \cup \{\perp\}).$$

Note that the symbol $\perp$ is used to represent the "undefined" value.

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. 3(a)) which refer to constructs with a singular primary key.
- Plural relationships (fig. 3(b)) which refer to constructs with a composite primary key.
- Functional relationships (fig. 3(c)) representing referential integrity constraints between entities and/or plural relationships.
- Mandatory relationships (fig. 3(d)) representing existence dependency constraints between entities and/or plural relationships.



(a) Operational Entity (b) Plural

(c) Functional (d) Mandatory

Figure 3: Structural Model Constructs

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 3NF.

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.

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. Section 4.1 will describe how TSER is used to produce a data model for scenario integration.

2.3 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, the Forspec team decided to use a real-world system to support its work. A University library system was decided upon and part of Université de Montréal's library system was investigated. It consisted of six scenarios, document borrowing, document return (two scenarios), borrowing extension, reader registration, and document registration. The average FSA for a scenario, has six states, seven entries, and ten transitions. A complete description of the example can be found in [11]. However, for presentation purposes, a scaled down library example will be used.

The Simple Library system involves three scenarios, document borrowing, document return, reader registration. In the borrowing scenario, a reader can borrow one or several documents. Also, the user can go directly from the borrowing scenario to any of the two other scenarios. In the return scenario, the reader may return one or several documents. The user may also switch directly to the borrowing scenario. In the reader registration scenario, a potential reader is registered and provided with a reader card. From there on, it is possible to register another reader, or to go to the borrowing scenario. A complete description of the borrowing scenario is provided in Section 3.1 below.

3 Scenario Acquisition and Behavior Modeling

3.1 Definition of Scenario

There are many definitions and uses of the scenario concept ([5, 6, 8, 10, 11, 16, 20]). 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 [18], and used in [1, 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 1 : the IS is a Banking system. A business task is to perform a banking transaction. The user is a customer of the bank.

Example 2 : 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, for performing the business task;
- a set of interactions (user action, expected CIS reaction(s)). An interaction accomplishes a token-task of the business task;
- the partial or complete ordering of the interactions;
- those interactions which end the scenario.

Several aspects of the definition are worth mentionning. All elements contributing to the definition of a scenario are provided by the user, or elicited from the user. Also, a scenario is more complex than a single sequence of interactions. Finally, all aspects of the definition are provided by the user in terms of the IS and of the business task. A scenario definition is elicited from the user(s) and results in an informal description. The narrative in Figure 4 for example, describes the Document borrowing scenario of the library example.

- 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 4: Description of the Document Loan Scenario

3.2 Formal Scenario Model

Previous works ([8]) have shown that it is possible to formalize the process and the product of scenario formalization. In this work, the formal model of a scenario will be assumed to be available.

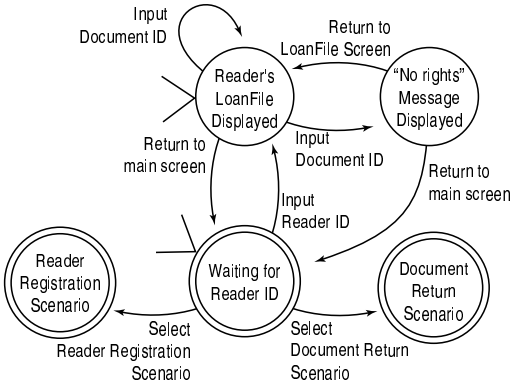
The behavioral part of a scenario will be modeled by a state-event automaton, a variation of a finite state machine, defined as follows

$$Mc = (Sc, Hc, Ic, Fc, Tc)$$

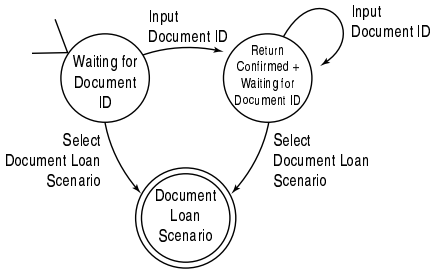
where :

- 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 (see scenario definition in Section 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.
- $Tc \subset Sc \times Hc \times Sc$ 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'.

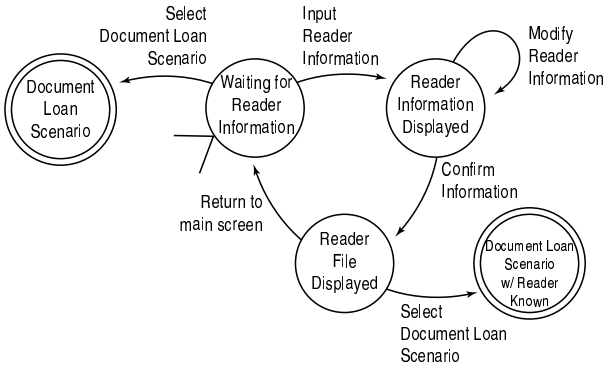
A state-event automaton can be represented by a state-event diagram. Figure 5 presents the state-event diagrams for the three scenarios of the Simple Library system described in Section 2.3.



(a) Document Loan



(b) Document Return



(c) Reader Registration

Figure 5: Transition Systems for the Scenarios

4 Scenario Analysis

In the Scenario Analysis step, performed on a scenario by scenario basis, data and other semantic information will be elicited, and then tied to the scenario automaton. The first activity, data modeling produces a formal model of the data entities visible in the scenario. These partial models are integrated in the second step. In the third activity, business rules and pre- and postconditions are elicited. Finally, entity states are constructed and tied to the appropriate scenario states. At the end of the scenario analysis step, all informations required for scenario integration will be available, that is, entities and entity states, business rules, and correspondence between entity states and scenario states.

4.1 Modeling Data

Data modeling consists mostly in (1) identifying the "objects" pertaining to a specific scenario, (2) the attributes describing these "objects", (3) the domain of these attributes, and (4) the functional dependencies among these attributes. The analysis is based on the informal and formal descriptions of the scenario. Figure 6 presents the resulting functional model for the Document Loan Scenario. Specifically, we identified three "objects", namely, Document which represents the different documents stored on the library shelves, LoanFile which represents the information about document loans, and Reader which represents the information about library users. For each of these "objects", we also identified attributes used in the scenario description, their respective domain, and the functional dependencies among them.

The next task is to create a normalized data model for each scenario. This is achieved by applying TSER's decomposition, normalization and consolidation steps to each scenario's functional models, as described in Section 2.2.

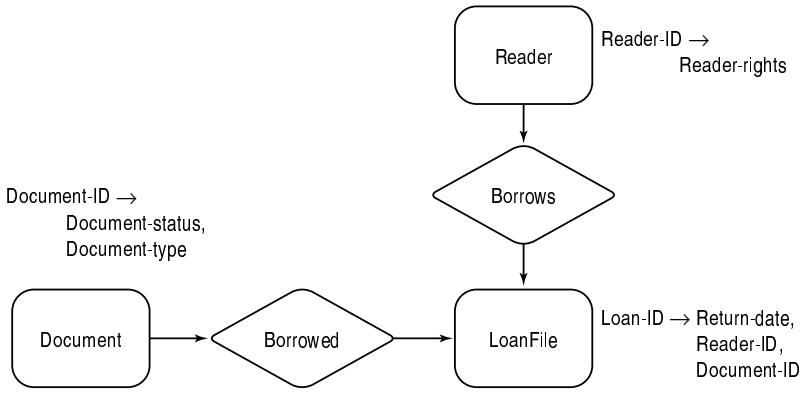
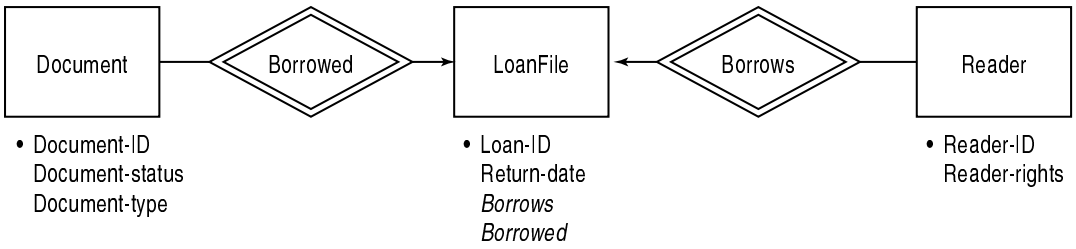


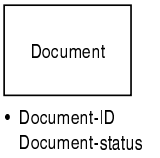
Figure 6: Functional Model for Document Loan Scenario

The structural models for the three scenarios resulting from this task are presented in Figure 7. Figure 8 provides detailed information on the attribute domaines, by entity and scenario.

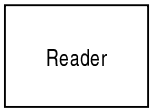
At this point, we have identified all the entities[1] used in each scenario. The Document Loan Scenario uses three entities (Document, LoanFile, and Reader), the Reader Registration Scenario only uses the Reader entity, and the Document Return Scenario only uses the Document entity.



(a) Document Loan



(b) Document Return



(c) Reader Registration

Figure 7: Structural Models for the Scenarios

4.2 Integrating Data Models

At this point, we produce an integrated data model using TSER's consolidation step from the normalized data models obtained for each scenario (Sect. 4.1). The resulting model for the Library example is illustrated in Figure 9. Figure 10 provides the integrated attribute domains.

We must point out that entities having the same key are merged into a single entity. Furthermore, attributes' domains are merged to obtain their complete domain.

[1] from this point on, we will refer to an operational entity or a plural relationship as an "entity"

Document :	Document-ID :	$code1$ where $code1$ is a set of IDs for Documents
	Document-status :	$\{available, loaned\}$
	Document-type :	$\{1, 2\}$
Reader :	Reader-ID :	$\{code2, \perp\}$ where $code2$ is a set of IDs for Readers
	Reader-rights :	$\{all\_documents, 1\_only, \perp\}$
LoanFile :	Loan-ID :	$code3$ where $code3$ is a set of IDs for LoanFiles
	Return-date :	$dates$ where $dates$ is a set of dates
	Borrows :	$code2$ (the domain is made of reader's Ids)
	Borrowed :	$code1$ (the domain is made of document's Ids).

(a) Document Loan Scenario

Document :	Document-ID :	$code1$
	Document-status :	$\{available, loaned\}$

(b) Document Return Scenario

Reader :	Reader-ID :	$\{code2, \perp\}$
	Reader-data :	$\{strings, \perp\}$ where $strings$ is a set of strings
	Reader-rights :	$\{all\_documents, 1\_only, \perp\}$

(c) Reader Registration Scenario

Figure 8: Attribute Domains by Entity and Scenario

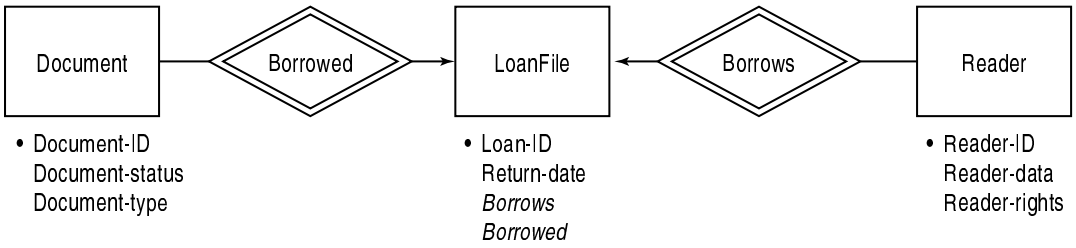


Figure 9: Library Integrated Structural Model

4.3 Eliciting Scenario Semantics

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

From an application's point of view, business rules define how things should be done. In banking for example, "no withdrawal on a negative balance" is a business rule. In the academic world, "no transcript to a student who has not paid his fees" is another example of a business rule. From a requirements engineering point of view, business rules 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 semantic, and will be used to construct the specification. Some of the business rules are implicit in the data model generated with TSER [9] (data integrity rules).

We define a special type of business rule, the existential rule for an entity. It is obvious that when an entity does not exist, all its attributes have the $\perp$ value. The existential rule specifies which attributes must be defined when an entity comes into existence.

Document :	Document-ID :	$code1$
	Document-type :	$\{1, 2\}$
	Document-status :	$\{available, loaned\}$
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 10: Attribute Domains for the Integrated Data Model

In the Library system the following rules have been identified.

Existential rules for the Document entity

Business Rule r_1 **:** If the entity does not exist, all its attributes do not exist

$$\text{Document-ID} = \perp \rightarrow \text{Document-status} = \perp \wedge \text{Document-type} = \perp$$

Business Rule r_2 **:** If the Reader exists, all its attributes are defined

$$\text{Document-ID} \in code1 \rightarrow (\text{Document-status} = available \vee \text{Document-status} = loaned) \wedge (\text{Document-type} = 1 \vee \text{Document-type} = 2)$$

Existential rules for the Reader entity

Business Rule r_3 **:** If the entity does not exist, all its attributes do not exist

$$\text{Reader-ID} = \perp \rightarrow \text{Reader-data} = \perp \wedge \text{Reader-rights} = \perp$$

Business Rule r_4 **:** If the Reader exists, all its attributes are defined

$$\text{Reader-ID} \in code2 \rightarrow \text{Reader-data} \in strings \wedge (\text{Reader-rights} = all\_documents \vee \text{Reader-rights} = 1\_only)$$

Existential rules for the LoanFile entity

Business Rule r_5 **:** If the entity does not exist, all its attributes do not exist

$$\text{Loan-ID} = \perp \rightarrow \text{Borrows} = \perp \wedge \text{Borrowed} = \perp \wedge \text{Return-date} = \perp$$

Business Rule r_6 **:** if the entity exists, all its attributes are defined

$$\text{Loan-ID} \in code3 \rightarrow \text{Borrows} \in code2 \wedge \text{Borrowed} \in code1 \wedge \text{Return-date} \in dates$$

Other business rules

Business Rule r_7 **:** If Reader is not registered, he (she) cannot borrow the document, and no LoanFile can exist for him (her)

$$\text{Reader-ID} = \perp \rightarrow \text{Loan-ID} = \perp \wedge \text{Borrows} = \perp \wedge \text{Borrowed} = \perp \wedge \text{Document-status} = available$$

Business Rule r_8 **:** If a document is in a LoanFile, the document's status must be "loaned"

$$\text{Borrowed} = \text{Document-ID} \rightarrow \text{Document-status} = loaned$$

Business Rule r_9 **:** If Reader has a LoanFile, its reading rights must be compatible with the type of the loaned document :

$$\text{Borrows} = \text{Reader-ID} \wedge \text{Borrowed} = \text{Document-ID} \rightarrow \text{Reader-rights} = all\_documents \vee (\text{Reader-rights} = 1\_only \wedge \text{Document-type} = 1)$$

4.3.2 Preconditions and Postconditions of Transitions

Preconditions and postconditions are business rules. Preconditions specify under which conditions a transition may be executed, while postconditions specify the effect of the transition on the entities. If a transition has no effect on entities, like a simple scenario state change for example, the precondition is also the postcondition of the transition. Preconditions and postconditions are usually expressed as boolean expressions of conditions on the values of entities. We will use $pre(tc)$ and $post(tc)$ to respectively represent the pre- and postcondition of a scenario transition. Table 1 defines the pre- and postconditions for the Document Loan scenario, transition by transition.

4.3.3 Truth Table Representation of Pre- and Postconditions

For automation purposes, the precondition of a transition can also be represented by a truth table. The columns correspond to entities involved in the condition and the lines are all the combinations of entity values satisfying the precondition. The same is done for the postcondition of a transition.

4.3.4 Processing Associated with Transitions

Some transitions like tc_5 in the Document Loan scenario involve entity processing. For these transitions, the associated processing has to be specified. Transition-associated processing will be 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, $pre(pc) = pre(tc)$ and $post(pc) = post(tc)$. The postcondition specifies the entity values resulting from the processing.

In the Document Loan scenario, only transition tc_5 involves entity processing. The specification of the transition processing is made of (pc_5 is for processing involved in tc_5) :

$pre(pc_5)$: 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 $\in$ *code*3

5 Scenario Integration

The goal of scenario integration is to produce an integrated and coherent specification of the system. In this section, we first describe what constitutes the formal system specification. The description of the tasks required to obtain that specification follows. We have identified three steps to obtain such a system specification : integration of data and behavior, system states generation, and system transitions generation. The scenario integration task will produce an integrated data model and a description of the system behavior, using a transition system.

5.1 Formal System Specification

For expressing the external specification of the system, a guarded sequential machine model is used. The system specification SY is as follows :

$$SY = (E, Mc_i, Sy, Hy, Py, Ty, Iy, Fy, R)$$

where

- E is the set of entities of the system,
- Mc_i is the set of scenario automata for all scenarios $c_1, c_2, ..., c_n$,
- Sy is the set of system states,
- Hy is the set of external events,
- Py is the set of transition-associated processing,
- $Ty \subset Sy \times Hy \times Py \times Sy$ is the set of transitions,
- Iy is the set of initial states,
- Fy is the set of final states,
- R is the set of business rules.

Table 1: Preconditions and post conditions of the Book Loan Scenario, by transition

Transition	Starting state	Event	Ending state	$pre(tc)$ / $post(tc)$
tc_1	Waiting for Reader ID (sc_{11})	Input Reader ID (hc_1)	Reader's LoanFile Displayed (sc_{14})	Pre Reader-ID $\in code2$ $Post$ Reader-ID $\in code2$
tc_2	Waiting for Reader ID (sc_{11})	Select Document Return Scenario (hc_2)	Document Return Scenario (sc_{12})	Pre NIL $Post$ NIL
tc_3	Waiting for Reader ID (sc_{11})	Select Reader Registration Scenario (hc_3)	Reader Registration Scenario (sc_{13})	Pre Reader-ID $= \perp$ $Post$ Reader-ID $= \perp$
tc_4	Reader's LoanFile Displayed (sc_{14})	Return to main screen (hc_4)	Waiting for Reader ID (sc_{11})	Pre NIL $Post$ NIL
tc_5	Reader's LoanFile Displayed (sc_{14})	Input Document ID (hc_5)	Reader's LoanFile Displayed (sc_{14})	Pre Document-status $= available \wedge$ (Reader-rights $= all\_documents \vee$ Document-type $= 1$) $Post$ Document-status $= loaned \wedge$ Borrows $=$ Reader-Id $\wedge$ Borrowed $=$ Document-ID $\wedge$ Loan-ID $\in code3$
tc_6	Reader's LoanFile Displayed (sc_{14})	Input Document ID (hc_5)	"No Rights" Message Displayed (sc_{15})	Pre Reader-rights $= 1\_only \wedge$ Document-type $= 2 \wedge$ Document-status $= available$ $Post$ Reader-rights $= 1\_only \wedge$ Document-type $= 2 \wedge$ Document-status $= available \wedge$ Loan-ID $= \perp$
tc_7	"No rights" Message Displayed (sc_{15})	Return to main screen (hc_4)	Waiting for Reader ID (sc_{11})	Pre NIL $Post$ NIL
tc_8	"No rights" Message Displayed (sc_{15})	Return to LoanFile Screen (hc_6)	Reader's LoanFile Displayed (sc_{14})	Pre NIL $Post$ NIL

5.1.1 Integration Baseline

The integration baseline is the set of results obtained in the preceding stages. The system specification SY will be formally derived from that baseline which is composed of :

- entities as defined in the Data Models Integration step (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 4.3.4.

5.1.2 Formal and Automatic Production of SY from the baseline

The remaining elements of SY are produced formally from the baseline.

- Sy is derived from the entities, transitions pre- and postconditions, and from the scenario states, all available in the baseline. The algorithms used, described in Sections 5.2 and 5.5, are formal and can be automated.
- $Hy = \bigcup Hc_i$, 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).
- $Py = \bigcup Pc_i$, where Pc_i is the set of processes of scenario c_i, directly part of the baseline.
- Ty is derived from scenario transitions and scenario states, available in the baseline, from system states, entities and their states. The algorithm, described in Section 5.6, is formal and can be completely automated.
- Iy is a by-product of system state production; an initial system state is compatible with a scenario initial state.
- Fy is a by-product of system state production; a system final state is compatible with a scenario final state.

5.2 A Formal Model for Integrating Data and Behavior

In this section, scenario states and entitites will be related by a formal model based on relations. In addition, an algorithm will be proposed for constructing entity states at the scenario level.

5.2.1 Definitions

Entity State Let e be an entity whose set of attributes is $A = \{a_1, a_2, ..., a_n\}$ and key attributes are $a_1, ..., a_k$ $(k \leq n)$;

Definition 1 *The entity domain, noted $Dom(e)$ is defined as follows :*

$$Dom(e) \triangleq Dom(a_1) \times ... \times Dom(a_k) \times Dom_{\perp}(a_{k+1}) \times ... \times Dom_{\perp}(a_n).$$

$Dom(e)$ can be written :

$$Dom(e) = sn_1 \cup ... \cup sn_m \text{ with } sn_j \subset Dom(e).$$

Definition 2 *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 3 *A state sn of entity e is compatible with state sc of scenario c if, for all vn_i in sn, at least one of the following conditions is satisfied :*

- *there exists a transition tc starting at sc, and vn_i satisfies $pre(tc)$;*
- *there exists a transition tc' ending at sc, and vn_i satisfies $post(tc')$.*

Definition 4 *A state sn of entity e is compatible with state sc of scenario c if there exists an entity e', having a state sn' such that :*

- *sn' is compatible with sc, as stated in Definition 3 above;*
- *There exists a business rule stating that e can be in state sn only when e' is in state sn'*

Definition 5 *A state sn of entity e is compatible with state sc of scenario c if e satisfies the following conditions :*

- *it is involved in no transition starting or ending at sc;*
- *it is involved in no business rule as stated in Definition 4 above;*
- *it is invisible in scenario state sc, and all its states are compatible with c.*

5.2.2 An Equivalence Relation on Entity States

We must first observe that the definition of compatibility given above can apply to a single element vn of $Dom(e)$. Let us consider a system defined by scenarios $c_1, c_2, ..., c_n$. Scenario c_i has scenario states $sc_{i1}, sc_{i2}, ..., sc_{im_i}$. Let e be an entity of the system and let $Dom(e)$ be the domain of values of e, as stated in Definition 2.

Relation on $Dom(e)$.

- Let Sc be any set of scenario states of the system.
- Let vn_i, vn_j, and vn_k be elements of $Dom(e)$;
- Let $Sc_i \subset Sc$ be the set of scenario states with which vn_i is compatible;
- Let $Sc_j \subset Sc$ be the set of scenario states with which vn_j is compatible;
- Let $Sc_k \subset Sc$ be the set of scenario states with which vn_k is compatible.

We define the relation Q such that

$$vn_i Q vn_j \text{ iff } Sc_i = Sc_j$$

It is easy to show that Q is an equivalence relation :

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

Q is symetric : if $vn_i Q vn_j$, then $Sc_i = Sc_j$; therefore, $Sc_j = Sc_i$, which implies $vn_j Q vn_i$;

Q is transitive : $vn_i Q vn_j$ implies $Sc_i = Sc_j$; $vn_j Q vn_k$ implies $Sc_j = Sc_k$; set equality ($=$) is transitive, therefore $Sc_i = Sc_k$, hence $vn_i Q vn_k$.

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 entity e.

Compatible set Let $Sn(e) = \{sn_1, ..., sn_n\}$ be the set of states of e as defined above. For each sn_i, there is an associated set of scenario states $Comp(sn_i)$ with which all values in sn_i are compatible. $Comp(sn_i)$ is defined as the compatible set of sn_i.

Empty Compatible Set There is no constraint on the content of $Comp(sn_i)$. It could be empty. One equivalence class can therefore correspond to those values of $Dom(e)$ which are compatible with no scenario state of Sc.

5.2.3 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 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 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 respect 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 respect 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 $vn_k \in sn$, all elements of sn are common to sn_i and sn'_j.

5.3 Construction of Entity States at the Scenario Level

Constructing entity states at the scenario level involves three steps :

1. defining the pre- and postconditions at each scenario state;
2. for each entity, defining the values compatible with each scenario state;
3. for each entity, calculating the entity states.

5.3.1 Preconditions and postconditions at a Scenario State

The Condition at a scenario state is constructed using the following rules. Let us consider a transition $tc = (sc, hc, sc')$.

Rule 1 : the precondition of transition tc becomes a precondition of scenario state sc;

Rule 2 : the postcondition of transition tc becomes a postcondition of scenario state sc';

Rule 3 : all preconditions, postconditions of a scenario state are considered as combined by the "$\vee$" operator;

Rule 4 : if transition tc has NIL as a precondition, the condition of scenario state sc is considered to be the implicit precondition of transition tc;

Rule 5 : if transition tc has NIL as postcondition, the precondition, explicit or implicit, of transition tc, becomes the postcondition of tc;

Rule 6 : For any entry state of the scenario, include as conditions the conditions required by the business task.

Because of Rule 5, iterations may be required in order to find the complete set of conditions at a given state. The order in which to process the transitions is defined as follows :

- process first all transitions with explicit preconditions, starting at an entry state of the scenario, and following a path to a final state;
- consider then the transitions with no explicit preconditions and postconditions, and process them in an order such as to minimize the number of iterations.

Such an approach, however is difficult to implement. A simpler approach is to calculates the transitive closure for all the scenario states.

Table 2 shows for each scenario state of the Document Loan Scenario the contribution of the transitions (see Table 1 for the meaning of abbreviated names).

Transitions tc_1, tc_4, tc_7 and tc_8 have no pre- nor postconditions. In order to avoid iterations, they will be processed in the order tc_8, tc_7, tc_4, tc_2. For tc_8, all the conditions at sc_{15} will be added to the conditions of sc_{14}. Then, for tc_7, the conditions at sc_{15} will be added to the conditions at sc_{11}. Then, for tc_4, the conditions at sc_{14} (as incremented) will be added to the conditions at sc_{11}. Finally, for tc_4, the conditions at sc_{11} (as incremented), will be added to the conditions at sc_{12}. The final result, i.e., the conditions at each scenario state are shown in Table 3.

5.3.2 Entity values compatible with a scenario state

In order to determine the entity state / scenario state compatibility, the following steps are performed for each entity :

1. Find the contributing attributes of the entity. An attribute is contributing to the entity states if any of the following holds :
   - it appears in a pre- or postcondition of some state of the scenario;
   - it is constrained by a business rule in which the left part contains an attribute appearing in a pre- or postcondition of some state.
2. Construct the truth table of the contributing attributes.
3. Eliminate the values incompatible with business rules.
4. Find values compatible with the scenario state :
   - a value compatible with some pre- or postcondition of the scenario state (corresponds to Definition 3 in Section 5.2);
   - a value constrained by a business rule in which the left part contains a value appearing in a pre- or postcondition of the scenario state (corresponds to Definition 4 in Section 5.2);

Table 2: Pre- and postconditions at each scenario state, by transition

Scenario state	Transition	Precondition	Postcondition
sc_{11}	Scenario precondition :		Document-status = *available* ∧ Loan-ID = ⊥
	tc_1 :	Reader-ID ∈ *code*2	Not applicable
	tc_2 :	NIL	Not applicable
	tc_4 :	Not applicable	NIL
	tc_7 :	Not applicable	NIL
	tc_3 :	Reader-ID = ⊥	Not applicable
sc_{12}	tc_2 :	Not applicable	NIL
sc_{13}	tc_3 :	Not applicable	Reader-ID = ⊥
sc_{14}	tc_5 :	Document-status = *available* ∧ (Reader-rights = *all_documents* ∨ Document-type =1)	Document-status = *loaned* ∧ Borrows = Reader-ID ∧ Borrowed = Document-ID ∧ LoanID ∈ *code*3
	tc_4 :	NIL	Not applicable
	tc_6 :	Reader-rights = 1*_only* ∧ Document-type = 2 ∧ Document-status = *available*	Not applicable
	tc_1 :	Not applicable	Reader-ID ∈ *code*2
	tc_8 :	Not applicable	NIL
sc_{15}	tc_7 :	NIL	Not applicable
	tc_8 :	NIL	Not applicable
	tc_6 :	Not applicable	Document-status=*available* ∧ Reader-rights = 1*_only* ∧ Document-type = 2 ∧ Loan-ID = ⊥

Table 3: Conditions at each scenario state

Scenario state	Conditions
Waiting for Reader ID (sc_{11})	Reader-ID $\in$ *code*2 $\vee$ Reader-ID $= \perp \vee$ (Document-status $=$ *available* $\wedge$ Loan-ID $= \perp$) $\vee$ (Document-status $=$ *available* $\wedge$ (Reader-rights $=$ *all_documents* $\vee$ Document-type $= 1$)) $\vee$ (Document-status $=$ *available* $\wedge$ Reader-rights $= 1\_only \wedge$ Document-type $= 2 \wedge$ LoanID $= \perp$) $\vee$ (Document-status $=$ *loaned* $\wedge$ Borrows $=$ Reader-ID $\wedge$ Loan-ID $\in$ *code*3)
Document Return Scenario (sc_{12})	Reader-ID $\in$ *code*2 $\vee$ Reader-ID $= \perp \vee$ (Document-status $=$ *available* $\wedge$ Loan-ID $= \perp$) $\vee$ (Document-status $=$ *available* $\wedge$ (Reader-rights $=$ *all_documents* $\vee$ Document-type $= 1$)) $\vee$ (Document-status $=$ *available* $\wedge$ Reader-rights $= 1\_only \wedge$ Document-type $= 2 \wedge$ LoanID $= \perp$) $\vee$ (Document-status $=$ *loaned* $\wedge$ Borrows $=$ Reader-ID $\wedge$ Loan-ID $\in$ *code*3)
Reader Registration Scenario (sc_{13})	Reader-ID $= \perp$
Reader's LoanFile Displayed (sc_{14})	(Document-status $=$ *available* $\wedge$ (Reader-rights $=$ *all_documents* $\vee$ Document-type $= 1$) $\vee$ (Document-status $=$ *loaned* $\wedge$ Borrows $=$ Reader-ID $\wedge$ Borrowed $=$ Document=ID $\wedge$ Loan-ID $\in$ *code*3) $\vee$ Reader-ID $\in$ *code*2 $\vee$ (Document-status $=$ *available* $\wedge$ Reader-rights $= 1\_only \wedge$ Document-type $= 2 \wedge$ Loan-ID $= \perp$)
"No rights" Message Displayed (sc_{15})	Document-status $=$ *available* $\wedge$ Reader-rights $= 1\_only \wedge$ Document-type $= 1 \wedge$ Loan-ID $= \perp$

- all values of the entity if it does not appears in cases above : the entity is invisible in that scenario state (corresponds to Definition 5 in Section 5.2).

Entity states are constructed by applying the *Entity State Generation Algorithm.*

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.

The *Entity State Generation Algorithm* applies this result in order to construct the entity states, based on the business rules identified.

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

Table 4: Truth Table of the Document Entity

Document-ID	Document-status	Document-type	Abbreviated name
$\perp$	$\perp$	$\perp$	vn_{11}
$\perp$	$\perp$	1	invalid (r_1)
$\perp$	$\perp$	2	invalid (r_1)
$\perp$	*available*	$\perp$	invalid (r_1)
$\perp$	*available*	1	invalid (r_1)
$\perp$	*available*	2	invalid (r_1)
$\perp$	*loaned*	$\perp$	invalid (r_1)
$\perp$	*loaned*	1	invalid (r_1)
$\perp$	*loaned*	2	invalid (r_1)
$\in code1$	$\perp$	$\perp$	invalid (r_2)
$\in code1$	$\perp$	1	invalid (r_2)
$\in code1$	$\perp$	2	invalid (r_2)
$\in code1$	*available*	$\perp$	invalid (r_2)
$\in code1$	*available*	1	vn_{12}
$\in code1$	*available*	2	vn_{13}
$\in code1$	*loaned*	$\perp$	invalid (r_2)
$\in code1$	*loaned*	1	vn_{14}
$\in code1$	*loaned*	2	vn_{15}

In the following subsections, we illustrates this algorithm for the Document, Reader, LoanFile entities, for the Document Loan Scenario.

5.3.3 Entity States of the Document Entity

Contributing attributes

- Mentioned in some scenario state condition : Document-status, Document-type.
- Involved in a relevant business rule : Document-ID, Document-status, Document-type.

Truth table See Table 4.

Elimination of values incompatible with business rules Most of the values have been eliminated, due to the existential rules of the entity. The remaining values have been assigned value identifiers in Table 4.

Values compatible with scenario states and basic partitions See Table 5.

States of the Document entity See Table 6.

The partitions of the Document entity are :

- $sn_{11} = \{vn_{12}\}$ (Document-status = *available* and Document-type = 1), compatible with scenario states sc_{11} to sc_{14},
- $sn_{12} = \{vn_{13}\}$ (Document-status = *available* and Document-type = 2), compatible with all states of the scenario,
- $sn_{13} = \{vn_{14}, vn_{15}\}$ (Document-status = *loaned*), compatible with scenario states sc_{11}, sc_{12}, and sc_{14},
- $sn_{14} = \{vn_{11}\}$ (Document Entity undefined), compatible with no scenario state.

Table 5: Basic partitions of the Document entity

Scenario state	Compatible values and Partition	Compatible sets
Waiting for Reader ID (sc_{11})	vn_{12}, vn_{13}, vn_{14}, vn_{15} $Sn(sc_{11}) =$ $\{\{vn_{12}, vn_{13}, vn_{14}, vn_{15}\}, \{vn_{11}\}\}$	$Comp(\{vn_{12}, vn_{13}, vn_{14}, vn_{15}\}) = \{sc_{11}\}$ $Comp(\{vn_{11}\}) = \emptyset$
Document Return Scenario (sc_{12})	vn_{12}, vn_{13}, vn_{14}, vn_{15} $Sn(sc_{12}) =$ $\{\{vn_{12}, vn_{13}, vn_{14}, vn_{15}\}, \{vn_{11}\}\}$	$Comp(\{vn_{12}, vn_{13}, vn_{14}, vn_{15}\}) = \{sc_{12}\}$ $Comp(\{vn_{11}\}) = \emptyset$
Reader Registration scenario (sc_{13})	vn_{12}, vn_{13} $Sn(sc_{13}) =$ $\{\{vn_{12}, vn_{13}\}, \{vn_{11}, vn_{14}, vn_{15}\}\}$	$Comp(\{vn_{12}, vn_{13}, vn_{14}, vn_{15}\}) = \{sc_{13}\}$ $Comp(\{vn_{11}, vn_{14}, vn_{15}\}) = \emptyset$
Reader's LoanFile Displayed (sc_{14})	vn_{12}, vn_{13}, vn_{14}, vn_{15} $Sn(sc_{14}) =$ $\{\{vn_{12}, vn_{13}, vn_{14}, vn_{15}\}, \{vn_{11}\}\}$	$Comp(\{vn_{12}, vn_{13}, vn_{14}, vn_{15}\}) = \{sc_{14}\}$ $Comp(\{vn_{11}\}) = \emptyset$
"No rights" Message Displayed (sc_{15})	vn_{13} $Sn(sc_{15}) =$ $\{\{vn_{13}\}, \{vn_{11}, vn_{12}, vn_{14}, vn_{15}\}\}$	$Comp(\{vn_{13}\}) = \{sc_{15}\}$ $Comp(\{vn_{11}, vn_{12}, vn_{14}, vn_{15}\}) = \emptyset$

Table 6: Construction of the states of the Document entity

(i, j)	Partition	Compatible sets
(1,1)	$Sn = \{\{vn_{12}, vn_{13}, vn_{14}, vn_{15}\}, \{vn_{11}\}\}$	$Comp(sn_1) = \{sc_{11}\}$ $Comp(sn_2) = \emptyset$
(1,2)	$Sn = \{\{vn_{12}, vn_{13}, vn_{14}, vn_{15}\}, \{vn_{11}\}\}$	$Comp(sn_1) = \{sc_{11}, sc_{12}\}$ $Comp(sn_2) = \emptyset$
(1,3)	$Sn = \{\{vn_{12}, vn_{13}\}, \{vn_{14}, vn_{15}\}, \{vn_{11}\}\}$	$Comp(sn_1) = \{sc_{11}, sc_{12}, sc_{13}\}$ $Comp(sn_2) = \{sc_{11}, sc_{12}\}$ $Comp(sn_3) = \emptyset$
(1,4)	$Sn = \{\{vn_{12}, vn_{13}\}, \{vn_{14}, vn_{15}\}, \{vn_{11}\}\}$	$Comp(sn_1) = \{sc_{11}, sc_{12}, sc_{13}, sc_{14}\}$ $Comp(sn_2) = \{sc_{11}, sc_{12}, sc_{14}\}$ $Comp(sn_3) = \emptyset$
(1,5)	$Sn = \{\{vn_{12}\}, \{vn_{13}\}, \{vn_{14}, vn_{15}\}, \{vn_{11}\}\}$	$Comp(sn_1) = \{sc_{11}, sc_{12}, sc_{13}, sc_{14}\}$ $Comp(sn_2) = \{sc_{11}, sc_{12}, sc_{13}, sc_{14}, sc_{15}\}$ $Comp(sn_3) = \{sc_{11}, sc_{12}, sc_{14}\}$ $Comp(sn_4) = \emptyset$

Table 7: Truth Table of the Reader entity

Reader-ID	Reader-rights	Reader-data	Abbreviated name
$\perp$	$\perp$	$\perp$	vn_{21}
$\perp$	$all\_documents$	$\perp$	invalid (r_3)
$\perp$	$all\_documents$	$\in strings$	invalid (r_3)
$\perp$	$\perp$	$\in strings$	invalid (r_3)
$\perp$	1_only	$\perp$	invalid (r_3)
$\perp$	1_only	$\in strings$	invalid (r_3)
$\in code2$	$\perp$	$\perp$	invalid (r_4)
$\in code2$	$\perp$	$\in strings$	invalid (r_4)
$\in code2$	$all\_documents$	$\perp$	invalid (r_4)
$\in code2$	$all\_documents$	$\in strings$	vn_{22}
$\in code2$	1_only	$\perp$	invalid (r_4)
$\in code2$	1_only	$\in strings$	vn_{23}

Table 8: Basic partitions of the Reader Entity

Scenario state	Compatible values and Partition	Compatible sets
Waiting for Reader ID (sc_{11})	vn_{21}, vn_{22}, vn_{23} $Sn(sc_{11}) = \{\{vn_{21}, vn_{22}, vn_{23}\}\}$	$Comp(\{vn_{21}, vn_{22}, vn_{23}\}) = \{sc_{11}\}$
Document Return Scenario (sc_{12})	vn_{21}, vn_{22}, vn_{23} $Sn(sc_{12}) = \{\{vn_{21}, vn_{22}, vn_{23}\}\}$	$Comp(\{vn_{21}, vn_{22}, vn_{23}\}) = \{sc_{12}\}$
Reader Registration scenario (sc_{13})	vn_{21} $Sn(sc_{13}) = \{\{vn_{21}\}, \{vn_{22}, vn_{23}\}\}$	$Comp(\{vn_{21}\}) = \{sc_{13}\}$ $Comp(\{vn_{22}, vn_{23}\}) = \emptyset$
Reader's LoanFile Displayed (sc_{14})	vn_{22}, vn_{23} $Sn(sc_{14}) = \{\{vn_{22}, vn_{23}\}, \{vn_{21}\}\}$	$Comp(\{vn_{22}, vn_{23}\} = \{sc_{14}\}$ $Comp(\{vn_{21}\} = \emptyset$
"No rights" Message Displayed (sc_{15})	vn_{23} $Sn(sc_{15}) = \{\{vn_{23}\}, \{vn_{21}, vn_{22}\}\}$	$Comp(\{vn_{23}\} = \{sc_{15}\}$ $Comp(\{vn_{21}, vn_{22}\} = \emptyset$

5.3.4 Entity States of the Reader Entity

Contributing attributes

- Mentioned in some scenario state condition : Reader-ID, Reader-rights.
- Involved in a relevant business rule : Reader-data.

Truth table See Table 7.

Elimination of values incompatible with business rules Most of the values have been eliminated, due to the existential rules of the entity. The remaining values have been assigned value identifiers in Table 7.

Values compatible with scenario states and basic partitions See Table 8.

States of the Reader entity See Table 9.

The partitions of the Reader entity are :

Table 9: Construction of the states of the Reader entity

(i,j)	Partition	Compatible sets
(1,1)	$Sn = \{\{vn_{21}, vn_{22}, vn_{23}\}\}$	$Comp(sn_1) = \{sc_{11}\}$
(1,2)	$Sn = \{\{vn_{21}, vn_{22}, vn_{23}\}\}$	$Comp(sn_1) = \{sc_{11}, sc_{12}\}$
(1,3)	$Sn = \{\{vn_{21}\}, \{vn_{22}, vn_{23}\}\}$	$Comp(sn_1) = \{sc_{11}, sc_{12}, sc_{13}\}$ $Comp(sn_2) = \{sc_{11}, sc_{12}\}$
(1,4)	$Sn = \{\{vn_{21}\}, \{vn_{22}, vn_{23}\}\}$	$Comp(sn_1) = \{sc_{11}, sc_{12}, sc_{13}\}$ $Comp(sn_2) = \{sc_{11}, sc_{12}, sc_{14}\}$
(1,5)	$Sn = \{\{vn_{21}\}, \{vn_{22}\}, \{vn_{23}\}\}$	$Comp(sn_1) = \{sc_{11}, sc_{12}, sc_{13}\}$ $Comp(sn_2) = \{sc_{11}, sc_{12}, sc_{14}\}$ $Comp(sn_3) = \{sc_{11}, sc_{12}, sc_{14}, sc_{15}\}$

Table 10: Truth Table of the LoanFile entity

Loan-ID	Borrows	Borrowed	Abbreviated name
$\perp$	$\perp$	$\perp$	vn_{31}
$\perp$	$\perp$	$\in code1$	invalid (Rule 1)
$\perp$	$\in code2$	$\perp$	invalid (Rule 1)
$\perp$	$\in code2$	$\in code1$	invalid (Rule 1)
$\in code3$	$\perp$	$\perp$	invalid (Rule 2)
$\in code3$	$\perp$	$\in code1$	invalid (Rule 2)
$\in code3$	$\in code2$	$\perp$	invalid (Rule 2)
$\in code3$	$\in code2$	$\in code1$	vn_{32}

- $sn_{21} = \{vn_{21}\}$ (Reader not registered), compatible with scenario states sc_{11}, sc_{12}, sc_{13},
- $sn_{22} = \{vn_{22}\}$ (Reader-ID $\in code2$, Reader-rights = $all\_documents$, Reader-data $\in strings$), compatible with scenario states sc_{11}, sc_{12}, and sc_{14},
- $sn_{23} = \{vn_{23}\}$ (Reader-ID $\in code2$, $Reader-rights = 1\_only$, Reader-data $\in strings$), compatible with scenario states sc_{11}, sc_{12}, sc_{14}, and sc_{15}.

5.3.5 Entity States of the LoanFile Entity

Contributing attributes

- Mentioned in some scenario state condition : Loan-ID, Borrows, Borrowed.
- Involved in a relevant business rule : none.

Truth table See Table 10.

Elimination of values incompatible with business rules Most of the values have been eliminated, due to the existential rules of the entity. The remaining values have been assigned value identifiers in Table 10.

Values compatible with scenario states and basic partitions See Table 11.

States of the LoanFile entity See Table 12

The states of the LoanFile entity are :

- $sn_{31} = \{vn_{31}\}$ (LoanFile does not exist), compatible with all scenario states,
- $sn_{32} = \{vn_{32}\}$ (LoanFile fully defined), compatible with scenario states sc_{11}, sc_{12}, sc_{14}.

Table 11: Basic partitions of the LoanFIle Entity

Scenario state	Compatible values and Partition	Compatible sets
Waiting for Reader ID (sc_{11})	vn_{31}, vn_{32} $Sn(sc_{11}) = \{\{vn_{31}, vn_{32}\}\}$	$Comp(\{vn_{31}, vn_{32}\}) = \{sc_{11}\}$
Document Return Scenario (sc_{12})	vn_{31}, vn_{32} $Sn(sc_{12}) = \{\{vn_{31}, vn_{32}\}\}$	$Comp(\{vn_{31}, vn_{32}\}) = \{sc_{12}\}$
Reader Registration scenario (sc_{13})	vn_{31} $Sn(sc_{13}) = \{\{vn_{31}\}, \{vn_{32}\}\}$	$Comp(\{vn_{31}\}) = \{sc_{13}\}$ $Comp(\{vn_{32}\}) = \emptyset$
Reader's LoanFile Displayed (sc_{14})	vn_{31}, vn_{32} $Sn(sc_{14}) = \{\{vn_{31}, vn_{32}\}\}$	$Comp(\{vn_{31}, vn_{32}\}) = \{sc_{14}\}$
"No rights" Message Displayed (sc_{15})	vn_{31} $Sn(sc_{15}) = \{\{vn_{31}\}, \{vn_{32}\}\}$	$Comp(\{vn_{31}\}) = \{sc_{15}\}$ $Comp(\{vn_{32}\}) = \emptyset$

Table 12: Construction of the states of the LoanFile entity

(i, j)	Partition	Compatible sets
1	$Sn = \{\{vn_{31}, vn_{32}\}\}$	$Comp(\{vn_{31}, vn_{32}\}) = \{sc_{11}, sc_{12}\}$
2	$Sn = \{\{vn_{31}\}, \{vn_{32}\}\}$	$Comp(\{vn_{31}\}) = \{sc_{11}, sc_{12}, sc_{13}\}$ $Comp(\{vn_{32}\}) = \{sc_{11}, sc_{12}\}$
3	$Sn = \{\{vn_{31}\}, \{vn_{32}\}\}$	$Comp(\{vn_{31}\}) = \{sc_{11}, sc_{12}, sc_{13}, sc_{14}\}$ $Comp(\{vn_{32}\}) = \{sc_{11}, sc_{12}, sc_{14}\}$
4	$Sn = \{\{vn_{31}\}, \{vn_{32}\}\}$	$Comp(\{vn_{31}\}) = \{sc_{11}, sc_{12}, sc_{13}, sc_{14}, sc_{15}\}$ $Comp(\{vn_{32}\}) = \{sc_{11}, sc_{12}, sc_{14}\}$

5.4 State Truth Table of Transition Pre- and Postconditions

The following is introduced to prove that scenario integration can be completely automated. Let $Tpre(tc)$ be the values truth table of transition tc (see Sect. 4.3.3), and let l be a line of that table. Let $e_1, e_2, ..., e_n$, be the entities involved in $Tpre(tc)$. Let $Sn(e_1), Sn(e_2), ...Sn(e_n)$, be respectively the set of states of entities $e_1, e_2, ..., e_n$.

Definition 6 *Entity states combination* $Sn(e_1) \times Sn(e_2) \times ... \times Sn(e_n)$ *satisfies* $pre(tc)$ *if* $l \in Sn(e_1) \times Sn(e_2) \times ... \times Sn(e_n)$.

Application The state truth table of $pre(tc)$ is obtained by replacing in $Tpre(tc)$ every line by the entity state combination which contains the line. In the resulting table, each line satisfies $pre(tc)$. The same is done for $post(tc)$.

5.5 Generating System States

The integrated data model has identified all the entities. Furthermore, The entity states were constructed based on the information available from all the scenarios. The next step is to generate the system states based on the entity states and the scenario states.

5.5.1 Obtaining System States

Potential System State A system state is characterized by the entities composing the system. Therefore, there must be a link between system state and entity state. Potentially, the system may be in any state the entities can be. A potential system state, noted w, is defined as one tuple in the carthesian product of all entity states. Given $E = \{e_1, ..., e_n\}$, a set of system entities, and $Sn(e)$, the set of integrated entity states for entity e, we have the set of potential system states W :

$$W \triangleq Sn(e_1) \times ... \times Sn(e_n).$$

Purified Potential System State Some of the potential system states are invalid, based on the semantics of the different scenarios, as identified in Section 4.3. Therefore, potential states must be "purified" to preserve only the states which agree with entity states constraints (i.e., business rules).

A purified potential system state, noted wp, is a potential system state which agrees with all the entity states constraints 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 :

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

System State

Definition 7 *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

Definition 8 *A system state sy is compatible with state sc of scenario c if, for all* wp_i *in sy, the following conditions is satisfied :*

$$sc \in Comp(wp)$$

where

$$Comp(wp) \triangleq \bigcap_{i=1}^{n} Comp(sn_i)$$

5.5.2 An Equivalence Relation on System States

We must first observe that the definition of compatibility given above can apply to a single element wp of sy. Let us consider a system defined by scenarios $c_1, c_2, ..., c_n$. Scenario c_i has scenario states $sc_{i1}, sc_{i2}, ..., sc_{im_i}$. Let Wp be the set of purified potential system states.

Relation on Wp.

- Let Sc be any set of scenario states of the system.
- Let wp_i, wp_j, and wp_k be elements of Wp;
- Let $Sc_i \subset Sc$ be the set of scenario states with which wp_i is compatible;
- Let $Sc_j \subset Sc$ be the set of scenario states with which wp_j is compatible;
- Let $Sc_k \subset Sc$ be the set of scenario states with which wp_k is compatible.

We define the relation Q' such that

$$wp_i Q' wp_j \text{ iff } Sc_i = Sc_j$$

It is easy to show that Q' is an equivalence relation :

Q' is reflexive : $wp_i Q' wp_i$ means that $Sc_i = Sc_i$;

Q' is symetric : if $wp_i Q' wp_j$, then $Sc_i = Sc_j$; therefore, $Sc_j = Sc_i$, which implies $wp_j Q' wp_i$;

Q' is transitive : $wp_i Q' wp_j$ implies $Sc_i = Sc_j$; $wp_j Q' wp_k$ implies $Sc_j = Sc_k$; set equality ($=$) is transitive, therefore $Sc_i = Sc_k$, hence $wp_i Q' wp_k$.

Being an equivalence relation on Wp, Q' induces a partition of Wp. The equivalence classes of the partition define the system states Sy.

Compatible set Let $Sy = \{sy_1, ..., sy_n\}$ be the set of system states as defined above. For each 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.

Empty Compatible Set There is no constraint on the content of $Comp(sy_i)$. It could be empty. One equivalence class can therefore correspond to those values of Wp which are compatible with no scenario state of Sc. These states represent invalid states in the context of current set scenario states Sc.

5.5.3 Integration Strategy

Although the approach described above yields a minimum number of system states, calculating these states may require $O(avg^{\|E\|} \cdot \| R \|)$, where

$$avg = \sum_{i=1}^{\|E\|} \frac{Sn(e)}{\| E \|}$$

is the average number of entity states per entity, $\| E \|$ is the number of states, and $\| R \|$ is the number of business rules.

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. Database queries usually consist of multiple carthesian products (or joins) with an equality constraint on certain values of the attributes. Optimization strategies will try to reduce, if not eliminate, the number of carthesian products required to obtain the query result [4].

This is accomplished by joining two relations (or partial results) at a time, and by using index information and query constraints to determine the best order to perform the different join operations. 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.

5.5.4 Integration Algorithms

Two algorithms are defined to generate system states (Sy) from entity states ($Sn(e)$). The first algorithm, the *Carthesian Product Sorting Algorithm*, is used to produce the carthesian product sequence. The second algorithm, the *System State Generation Algorithm*, produces the system states based on that carthesian product sequence, on the entity states, and on the business rules. This section presents these algorithms.

Carthesian Product Sorting Algorithm The Carthesian Product Sorting Algorithm constructs the carthesian product evaluation tree. As stated earlier, each leaf is the set of states for one entity. Each node is a carthesian product between two subtrees, constrained by some business rules. The tree structure is implicit, however, because we only determine the sequence in which each product in the tree should be performed.

The first product is identified by looking at the the number of potential system states produced when merging (i.e., applying a carthesian product) the states of two entities. The pair that yields the smallest number is merged first. The subsequent products are determined in the same manner. However, we do not consider the entities already merged, but rather the result from that merge operation. The algorithm performs in $O(\| E \|^4)$.

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} carthesian product;
Let $Ord_2(i)$ be the second set of entities being merged at the i^{th} carthesian 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
$Ord_1(i) \leftarrow E$

Applying the Carthesian Product Sorting Algorithm Table 13 illustrates the application of the Carthesian Product Sorting Algorithm to the Simple Library Case, limiting this study to the Document Loan Scenario. Based on this result, we first integrate entities Document and LoanFile. Then, we integrate this partial result with the Reader entity.

System States Generation Algorithm The *System States Generation Algorithm* generates the purified potential system states and removes invalid states as early as possible. It incrementally creates these states by following the carthesian product order determined by the *Carthesian Product Sorting Algorithm.* At each increment, we merge two previously obtained partial products.

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;

Table 13: Determining the Carthesian Product Order

Iteration	Variable	Value
$i = 0$	$o(\{e_1\})$	4
	$o(\{e_2\})$	3
	$o(\{e_3\})$	2
	$avg(e_1, 0)$	1
	$avg(e_2, 0)$	1
	$avg(e_3, 0)$	1
	Π	$\{\{e_1\}, \{e_2\}, \{e_3\}\}$
$i = 1$	$o(\{e_1, e_2\})$	12
	$o(\{e_1, e_3\})$	5
	$o(\{e_2, e_3\})$	6
	$avg(e_1, 1)$	1.25
	$avg(e_2, 1)$	0.4
	$avg(e_3, 1)$	1
	Π	$\{\{e_1, e_3\}, \{e_2\}\}$
	$Ord_1(1)$	$\{e_1\}$
	$Ord_2(1)$	$\{e_3\}$
$i = 2$	$o(\{e_1, e_2, e_3\})$	3
	$avg(e_1, 2)$	0.75
	$avg(e_2, 2)$	1
	$avg(e_3, 2)$	0.67
	Π	$\{\{e_1, e_2, e_3\}\}$
	$Ord_1(1)$	$\{e_1, e_3\}$
	$Ord_2(1)$	$\{e_2\}$

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} carthesian product resulting from the *Carthesian Product Sorting Algorithm*;
Let $Ord_2(i)$ be the second set of entities being merged at the i^{th} carthesian product resulting from the *Carthesian Product Sorting Algorithm*;
Let $W(\{e_1, ..., e_n\})$ be the purified potential system states in the carthesian product 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 $w' \in Ord_2(i)$
      $OK \leftarrow$ 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 on the set of compatible scenario states.

$Sy \leftarrow \emptyset$
For each $w \in W(E)$
$found \leftarrow$ False
For each $sy \in Sy$
If $Comp(w) = Comp(sy)$
$sy \leftarrow sy \cup \{w\}$
$found \leftarrow$ 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

Applying the System States Production Algorithm Table(14) shows the partial results of the carthesian products to obtain the purified potential system states. By partitioning Wp, we obtain the following system states :

- 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.

5.6 Generating External System Transitions

5.6.1 Definition and Properties

The specification considers only external transitions, corresponding to scenario transitions. A system transition is a tuple (sy, hy, py, sy'), such that there exists a scenario transition $tc = (sc_{ij}, hc, sc_{ik})$ in scenario c_i, and sy is a system state compatible with scenario state sc_{ij}, sy' is a system state compatible with scenario state sc_{ik}, $hy = hc$ is the event of scenario c_i, triggering scenario transition tc, py is the processing involved in scenario transition tc, specified by $pre(tc)$ and $post(tc)$.

Properties of System Transitions

Property 1 *Compatibility of system state and precondition of scenario transition. Let sy be a system state. Let* $Tpre(tc)$ *be the state truth table of scenario transition tc's precondition. If* $sy \cap Tpre(tc) \neq \emptyset$*, a system transition ty, corresponding to tc, may start at system state sy.*

Property 2 *Compatibility of system state and postcondition of scenario transition. Let sy' be a system state. Let* $Tpost(tc)$ *be the state truth table of scenario transition tc's postcondition. If* $sy' \cap Tpost(tc) \neq \emptyset$*, a system transition ty, corresponding to tc, may end at system state sy'.*

Table 14: Obtaining System States

Potential states	$Comp(w)$	Comments
$W(\{e_1\}) = \{$		
sn_{11}	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}\}$	
sn_{12}	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}, sc_{15}\}$	
sn_{13}	$\{sc_{11}, sc_{12}, sc_{14}\}$	
sn_{14}	$\emptyset$	Removed
$\}$		
$W(\{e_2\}) = \{$		
sn_{21}	$\{sc_{11}, sc_{12}, sc_{13}\}$	
sn_{22}	$\{sc_{11}, sc_{12}, sc_{14}\}$	
sn_{23}	$\{sc_{11}, sc_{12}, sc_{14}, sc_{15}\}$	
$\}$		
$W(\{e_3\}) = \{$		
sn_{31}	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}, sc_{15}\}$	
sn_{32}	$\{sc_{11}, sc_{12}, sc_{14}\}$	
$\}$		
$W(\{e_1, e_3\}) = \{$		
(sn_{11}, sn_{31})	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}\}$	
(sn_{11}, sn_{32})		Invalid (r_8)
(sn_{12}, sn_{31})	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}, sc_{15}\}$	
(sn_{12}, sn_{32})		Invalid (r_8)
(sn_{13}, sn_{31})	$\{sc_{11}, sc_{12}, sc_{14}\}$	
(sn_{13}, sn_{32})	$\{sc_{11}, sc_{12}, sc_{14}\}$	
$\}$		
$W(\{e_1, e_2, e_3\}) = \{$		
$(sn_{11}, sn_{21}, sn31)$	$\{sc_{11}, sc_{12}, sc_{13}\}$	
$(sn_{11}, sn_{22}, sn31)$	$\{sc_{11}, sc_{12}, sc_{14}\}$	
$(sn_{11}, sn_{23}, sn31)$	$\{sc_{11}, sc_{12}, sc_{14}\}$	
$(sn_{12}, sn_{21}, sn31)$	$\{sc_{11}, sc_{12}, sc_{13}\}$	
$(sn_{12}, sn_{22}, sn31)$	$\{sc_{11}, sc_{12}, sc_{14}, sc_{15}\}$	
$(sn_{12}, sn_{23}, sn31)$	$\{sc_{11}, sc_{12}, sc_{14}, sc_{15}\}$	
$(sn_{13}, sn_{21}, sn31)$		Invalid (r_7)
$(sn_{13}, sn_{22}, sn31)$	$\{sc_{11}, sc_{12}, sc_{14}\}$	
$(sn_{13}, sn_{23}, sn31)$	$\{sc_{11}, sc_{12}, sc_{14}\}$	
$(sn_{13}, sn_{21}, sn32)$		Invalid (r_7)
$(sn_{13}, sn_{22}, sn32)$	$\{sc_{11}, sc_{12}, sc_{14}\}$	
$(sn_{13}, sn_{23}, sn32)$	$\{sc_{11}, sc_{12}, sc_{14}\}$	
$\}$		

Serialisation (or non-concurrency) assumption. it will be assumed that at any time, at most one transition is executed by the system. At the implementation level, the assumption is reasonable, because it does not preclude concurrent executions of scenario transitions at the system level : if two scenario transitions are concurrently executed, the assumption results in an interleave of the executions of transitions belonging to different scenarios. The FSA model precludes already concurrent executions of transitions of the same scenario.

Definition 9 *Let $E = \{e_1, ..., e_n\}$ be the set of entities and $sy = \{w_1, ..., w_m\} \subset Sn(e_1) \times ... \times Sn(e_n)$ be a system state. The projection of a system state sy on entities $e'_1, ..., e'_{n'}$, $proj(sy, \{e'_1, ..., e'_{n'}\})$, is the set $\{w'_1, ..., w'_m\}$ where w'_i is constructed from w_i by only keeping values for entities $e'_1, ..., e'_{n'}$.*

Property 3 *Coherence between starting and ending state of a system transition. If the serialisation assumption holds when system transition ty corresponding to scenario transition tc is executed, only those entities involved in the transition may change. As a consequence, the entities not involved in post(tc) must be in the same states in sy and in sy'. This can be stated formally as follows.*

Let $e_1, e_2, ..., e_n$ be the entities not involved in post(tc). Tuple (sy, hy, py, sy') will be a system transition only if $proj(sy, \{e_1, e_2, ..., e_n\}) = proj(sy', \{e_1, e_2, ..., e_n\})$.

Given a scenario transition and a pair of system states, all three properties 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).

5.6.2 Application to Transition Construction

The algorithm will proceed for each scenario, scenario transition by scenario transition.

Algorithm 4 *Entity External Transition Generation Algorithm.*

Let $E' \subset E$ be a set of entities;
Let Ty be the set of all system transitions;
Let Tc be the set of all scenario transitions;
Let $tc = (sc_{ij}, hc, sc_{ik})$ be a scenario transition;
Let $Sy' \subset Sy$ be a set of system states;
Let $Sy'' \subset Sy$ be a set of system states;
Let hy be a system even;
Let py be a system process;
Let sy be a system state;
Let sy' be a system state.

$Ty \leftarrow \emptyset$
For each $tc \in Tc$

A- Construction of potential system transitions.

$Sy' \leftarrow$ the set of system states compatibles with sc_{ij}
$Sy'' \leftarrow$ the set of system states compatibles with sc_{ik}
$E' \leftarrow$ the set of entities not involved in tc

Sy' and Sy'' can be derived automatically from the compatible sets of system states (see Section 5.5 above).

B- Verification of Properties.
System states $sy \in Sy'$ and $sy' \in Sy''$ are candidates for being the starting and ending state, respectively, of a system transition.

For each $sy \in Sy'$
For each $sy' \in Sy''$
If $(sy \cap Tpre(tc) \neq \emptyset) \wedge (sy' \cap Tpost(tc) \neq \emptyset) \wedge (proj(sy, E') = proj(sy', E')$
$hy \leftarrow hc$
$py \leftarrow$ the process specified by $pre(tc)$ and $post(tc)$

Table 15: Obtaining System Transitions

Tc	Ty	Definition
tc_1	ty_1	(sy_2, hc_1, pc_1, sy_2)
	ty_2	(sy_3, hc_1, pc_1, sy_3)
tc_2	ty_3	(sy_1, hc_2, pc_2, sy_1)
	ty_4	(sy_1, hc_2, pc_2, sy_2)
	ty_5	(sy_1, hc_2, pc_2, sy_3)
tc_3	ty_6	(sy_1, hc_3, pc_3, sy_1)
tc_4	ty_7	(sy_1, hc_4, pc_4, sy_1)
	ty_8	(sy_2, hc_4, pc_4, sy_2)
	ty_9	(sy_3, hc_4, pc_4, sy_3)
tc_5	ty_{10}	(sy_2, hc_5, pc_5, sy_2)
	ty_{11}	(sy_3, hc_5, pc_5, sy_3)
tc_6	ty_{12}	(sy_3, hc_5, pc_6, sy_3)
tc_7	ty_{13}	(sy_3, hc_4, pc_7, sy_3)
tc_8	ty_{14}	(sy_3, hc_6, pc_8, sy_3)

$Ty \leftarrow Ty \cup \{(sy, hy, py, sy')\}$
End-If
End-For
End-For
End-For

At the end of the algorithm, all external system transition are defined. It is to be noted that a scenario transition may create several system transitions. Table 15 list the system transitions produced. The resulting system automaton is given in Figure 11. It should be noted that only transitions ty_9 $ty_1 3$ are redundant, since process pc_4 and pc_t are actually identical. Furthermore, it is obvious that some processing is missing from this system :

- No transition from sy_1 to the other states. A reader must register before he/she can borrow book. Therefore, the registration scenario is missing.
- No transition from sy_2 to sy_3. The return scenario has not been included in the generation process.

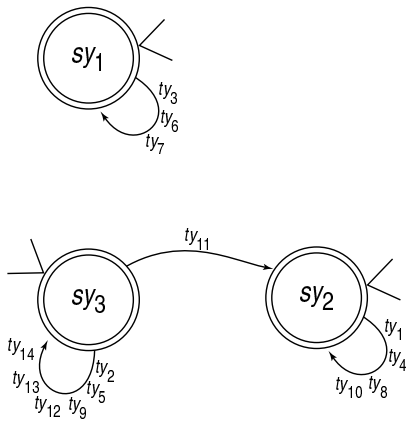


Figure 11: Final System Automaton

6 Incremental Integration

The approach we have presented in the previous sections supposed that the system specifications were obtained from all the scenarios describing the system. For instance, we assume that all the entity states are known prior

to system state generation. However, the algorithms could be modified to allow incremental integration. At each increment, we would model the new scenarios to consider using the approach presented in Section 4. This is easily done independantly from already analyzed scenarios. Furthermore, the generation of system states can start from an already existing partial specification.

7 Discussion/Conclusion

The overal objective of this work was to apply formal methods not only to the result but also to the process of requirements engineering. A major decision was to use the scenario concept. Our scenario definition was broader than many in the sense that it is not limited to a sequence of interactions. Moreover the definition is related to the system aims by the concept of business task. The formal tool used to model the scenario's behavior was quite conventional, a finite state machine which was enriched in the scenario analysis step. A major objective 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. Integration of data and behavior was defined formally at the scenario level and at the system level. At the scenario level, the concepts of entity states and of compatibility between entity state and scenario state, both formally defined, achieved complete semantic integration of data and behavior. The part of system semantic involved in the requirements was expanded by introducing the concept of business rules, formalized in first order predicate logic, and by using the rules to trim entity states of impossible or forbidden values. At the system level, the tie between data and behavior was also achieved by basing the definition of system states on combinations of entity states, and by basing the definition of system transitions on pre- and postconditions of scenario transitions. The first objective, using formal methods in the requirements elicitation process was achieved at the scenario integration stage. The baseline creation steps, all performed at the scenario analysis stage, are mostly manual, but provide the formal artifacts required by the automatic integration algorithm. And finally, the objective of controlling a potential combinatory explosion, was achieved by using well-tried methods drawn from database processing, and semantic information provided by the business rules.

Several decisions taken to achieve our objectives may be discussed. First the formal model used to specify the system. Any finite state-based model may be criticed for raising the risk of combinatorial explosion of states or transitions. Two safeguards were used in this work, namely applying database processing techniques, and using semantic information to trim the states produced by the integration algorithm. 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. The integration algorithm might be improved by making more use of the system semantic, for example by using the business rules to select which Cartesian product to perform first.

Our approach has some limitations. It handles only one instance of each entity and one instance of each scenario. Moreover, although no relative position of the scenarios is explicitely stated, 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 possibty 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] Gilbert Babin, François Lustman, and Peretz Shoval. Specification and design of transactions in information systems: A formal approach. *IEEE Transactions on Software Engineering*, 17(8):814–829, August 1991.

[2] Anne Dardenne. On the use of scenarios in requiremets acquisition. CIS-TR 93–17, Department of Computer Science, University of Oregon, Eugene, Oregon 97403, August 1993.

[3] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, April 1993.

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

[5] Jules Desharnais, Marc Frappier, Ridha Khédri, and Ali Mili. Integration of sequential scenarios. In *Sixth European Software Engineering Conference (ESEC'97)*, number 1301 in Lecture Notes in Computer Science, pages 310–326, Zurich, Switzerland, November 1997. Springer-Verlag. published as *Software Engineering Notes*, 22(6).

[6] Martin 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):254–271, 1995.

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

[8] Pei Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyohima, and C. Chen. Formal approach to scenario analysis. *IEEE Software*, 11:33–41, March 1994.

[9] Cheng Hsu, Yicheng Tao, M'hamed Bouziane, and Gilbert Babin. Paradigm translation in manufacturing information using a meta-model: The tser approach. *Ingénierie des systèmes d'information*, 1(3):325–352, January 1993.

[10] Ivar Jacobson, Magnus Christerson, Patrick Jonsson, and Gunnar Overgaard. *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley, revised edition, 1994.

[11] Ilka Kawashita. Spécification formelle des systèmes d'information par la technique des scénarios. Master's thesis, Département d'informatique et de recherche opérationnelle, Montréal, Québec, Canada, April 1997. François Lustman (Département d'informatique et de recherche opérationnelle, Université de Montréal).

[12] I. Khriss, M. Elkoutbi, and R. Keller. Automating the synthesis of statechart diagrams from multiple collaboration diagrams. In *Proc. International Workshop on the Unified Modeling Language "UML"'98 : Beyond the Notation*, pages 115–126bis, Mulhouse, France, June 1998.

[13] François Lustman. Specifying transaction-based information systems with regular expressions. *IEEE Transactions on Software Engineering*, 20(3):207–217, March 1994.

[14] François Lustman. A formal approach to scenario integration. *Annals of Software Engineerig*, 3:255–272, September 1997.

[15] Rational Software Corporation, Microsoft, Hewlett Packard, Oracle, Sterling, MCI, Unysis, ICON, IntelliCorp, I-Logix, IBM, ObjecTime, Platinum, Ptech, Taskon, Reich Technologies, and Softeam. UML notation guide, version 1.1, September 1997.

[16] K.S. Rubin and A. Goldberg. Object behavior analysis. *Communications of the ACM*, 35(9):48–62, September 1992.

[17] James Rumbaugh, Michel Blaha, Willian Premerlani, Frederick Eddy, and Willian Lorensen. *Object-oriented Modeling and Design*. Prentice-Hall, 1991.

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

[19] Stéphane Somé, Rachida Dssouli, and Jean Vaucher. Automation of requirements engineering using scenarios. Technical Report 978, Université de Montréal, Département d'informatique et recherche opérationnelle, University of Montreal, 1995.

[20] Stéphane Somé, Rachida Dssouli, and Jean Vaucher. From scenarios to timed automata: Building specifications from users requirements. In *2nd Asia Pacific Software Engineering Conference APSEC 95*, Brisbane, Australia, December 1995. ASPEC, 2nd Asia Pacific Software Engineering Conference APSEC 95.