Proceedings of the IASTED International Conference
Software Engineering and Applications (SEA '99)
October 6-8, 1999, Scottsdale, Arizona, USA

Gilbert Babin and François Lustman. "Formal Data and Behavior Requirements Engineering : A Scenario-based Approach." in 3rd Annual IASTED International Conference on Software Engineering Applications (SEA '99). Scottsdale, Arizona, USA. October 1999. pp. 119-125.

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

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

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

Abstract

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

Key Words — requirements engineering, formal methods, scenarios.

1 Introduction

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

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

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

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

2 Method Overview

2.1 Requirements Engineering

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

2.2 A Simple Library Example

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

Gilbert Babin and François Lustman. "Formal Data and Behavior Requirements Engineering : A Scenario-based Approach." in 3rd Annual IASTED International Conference on Software Engineering Applications (SEA '99). Scottsdale, Arizona, USA. October 1999. pp. 119-125.

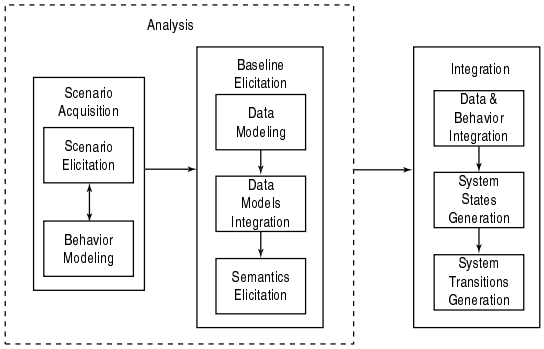

Fig. 1: Requirements Engineering Process

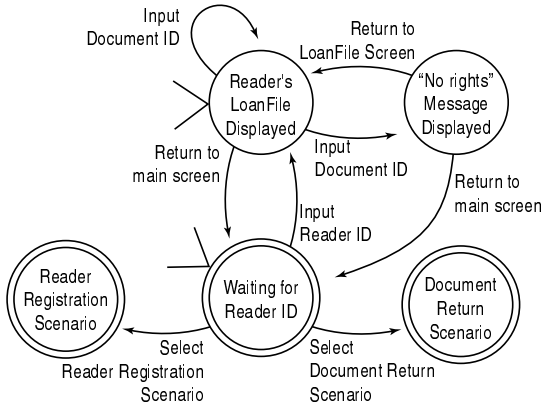

Fig. 2: Transition System for the Document Loan Scenario

here. The behavior part of the Document Loan scenario is described in Figure 2.

3 Scenario Acquisition

3.1 Definition of Scenario

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

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

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

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

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

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

3.2 Formal Scenario Model

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

- Sc is a set of states. The scenario is in a state when a system reaction is finished and the user has not entered an event.
- Hc is the set of events. An event is a gesture exercised by the user on the interface, indicating to the system that the user has finished entering all the elements of an action (Sect. 3.1). An event is labelled by the user-defined action.
- Ic is the set of initial states. Initial states correspond to user-defined entry points.
- Fc is the set of final states. Interactions defined by the user as ending the scenario, end on a final state.
- $Tc \subset Sc \times Hc \times Sc$ 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 2 presents the state-event diagram for the Document Loan scenario.

4 Baseline Elicitation

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

4.1 Data Modeling

Data modeling consists mostly in identifying (1) the "objects" 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. We rely on TSER [14] Methodology for these tasks. In the Document Loan scenario, we identified three "objects", namely, Document, LoanFile, and Reader. For each of these "objects", we also identified attributes used in the

Gilbert Babin and François Lustman. "Formal Data and Behavior Requirements Engineering : A Scenario-based Approach." in 3rd Annual IASTED International Conference on Software Engineering Applications (SEA '99). Scottsdale, Arizona, USA. October 1999. pp. 119-125.

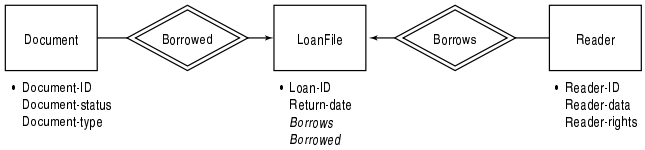

Fig. 3: Library Integrated Structural Model

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

Fig. 4: Attribute Domains for the Integrated Data Model

scenario description, their respective domain, and the functional dependencies among them.

The next task is to create a normalized data model (BCNF) for each scenario. This process breaks down the "objects" into normalized entities and relationships. For the purpose of this work, we will consider an entity any normalized construct having a primary key, either singular or composed. By repeating this process to each scenario, we identify all the entities used in each scenario.

4.2 Data Models Integration

At this point, we produce an integrated data model, from the normalized data models obtained for each scenario (Sect. 4.1). This task is automated using TSER's algorithms. The resulting model for the Library example is illustrated in Figure 3. Figure 4 provides the integrated attribute domains. A special domain value noted "$\perp$" represents the value "not defined."

4.3 Semantics Elicitation

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

4.3.1 Business Rules

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

Tab. 1: Business Rules in the Library System

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

4.3.2 Preconditions and Postconditions of Transitions

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

4.3.3 Processing Associated with Transitions

Some transitions involve entity processing. For these transitions, the associated processing has to be specified. 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 one transition (tc_5=(Reader's Loan File Displayed, Input Document ID, Reader's Loan File Displayed)) involves entity processing. The specification of the transition processing is made of (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 $\in code3$

4.4 Integration Baseline

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

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

5 Integration

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

5.1 Formal System Specification

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

The elements of SY are either directly found in, or produced formally from, the baseline as follows.

- Sy is derived from the entities, transitions pre- and postconditions, and from the scenario states, all available in the baseline. The algorithms used (see Sections 5.2 and 5.3) 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 5.4, 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 Data and Behavior Integration

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

5.2.1 Concept of Entity State

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

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

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

1. there exists a transition tc starting at sc, and vn_i satisfies $pre(tc)$;
2. there exists a transition tc' ending at sc, and vn_i satisfies $post(tc')$.
3. there exists an entity e', having a state sn' such that :
   - sn' is compatible with sc, as stated in Conditions 1 or 2 above;
   - there exists a business rule stating that e can be in state sn only when e' is in state sn'
4. the following conditions are all satisfied :
   - e is involved in no transition starting or ending at sc;
   - e is involved in no business rule as stated in Condition 3 above;
   - e is invisible in scenario state sc, and all its states are compatible with c.

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

It is easily shown that Q is an equivalence relation (see [15] for the proof). Being an equivalence relation on $Dom(e)$, Q induces a partition of $Dom(e)$. The equivalence classes of the partition define the states sn_i of e.

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

Tab. 2: Entity States for the Library System

State	Compatiblity set	Meaning
sn_{11}	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}\}$	Document-status = $available$ ∧ Document-type = 1
sn_{12}	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}, sc_{15}\}$	Document-status = $available$ ∧ Document-type = 2
sn_{13}	$\{sc_{11}, sc_{12}, sc_{14}\}$	Document-status = $loaned$
sn_{14}	$\emptyset$	Document-ID = $\perp$
sn_{21}	$\{sc_{11}, sc_{12}, sc_{13}\}$	Reader-ID = $\perp$
sn_{22}	$\{sc_{11}, sc_{12}, sc_{14}\}$	Reader-ID ∈ $code2$ ∧ Reader-rights = $all\_documents$ ∧ Reader-data ∈ $strings$
sn_{23}	$\{sc_{11}, sc_{12}, sc_{14}, sc_{15}\}$	Reader-ID ∈ $code2$ ∧ Reader-rights = 1_only ∧ Reader-data ∈ $strings$
sn_{31}	$\{sc_{11}, sc_{12}, sc_{13}, sc_{14}, sc_{15}\}$	LoanFile-ID = $\perp$
sn_{32}	$\{sc_{11}, sc_{12}, sc_{14}\}$	LoanFile-ID ∈ $code3$ ∧ Borrows ∈ $code2$ ∧ Borrowed ∈ $code1$

5.2.2 Construction of Entity States at the Scenario Level

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

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

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

The proof may be found in [15].

Construction of Entity States Entity States are constructed by applying the composition of relations property defined above, one scenario state at a time. The process involves three steps : (1) defining the pre- and postcondition at each scenario state; (2) for each entity, defining the values compatible with each scenario state; (3) for each entity, calculating the entity states.

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

5.3 System States Generation

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

5.3.1 System State Definition

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

Some of the potential system states are invalid, based on the business rules (Sect. 4.3). A *purified potential system state*, noted wp, is a potential system state which agrees with all the business rules identified in all the scenarios. Given $R = \{r_1, ..., r_m\}$, a set of business rules, we have the set of purified potential system states Wp, which is defined as $\{w = (sn_1, ..., sn_n) \in W \mid r_1 \wedge ... \wedge r_m\}$.

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

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

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

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

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

5.3.2 Integration Algorithm

Although the use of business rules minimizes the 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 entities identified in the system, 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 [16]. In RDBS, the "query plan" usually consists of a tree where the leaves are the relations to join to obtain the final query result, and the nodes are join operations to perform.

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

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

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

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

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

In the Library system, when only considering the Document Loan scenario, we obtain the following system states (see also Fig. 5) :

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

5.4 Generating External System Transitions

5.4.1 Definition and Properties

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

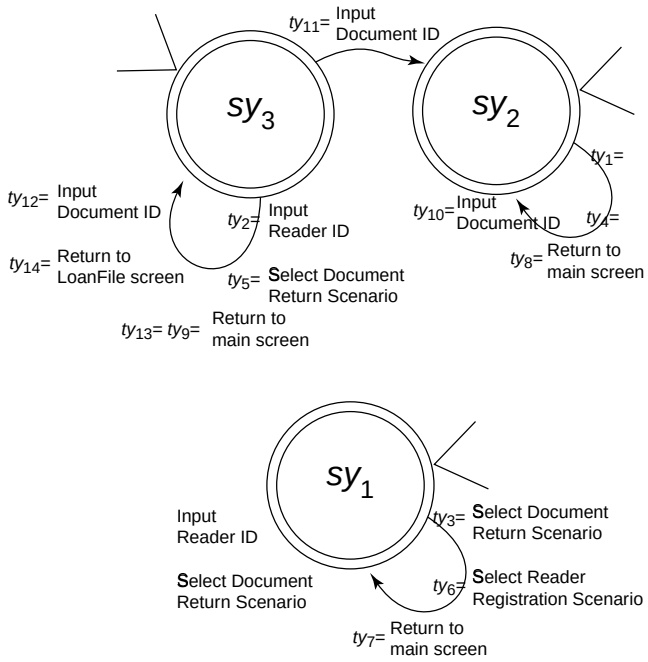

Fig. 5: Final System Automaton

because the required information is available either in the baseline (pre- and postconditions) or as result of the system state generation step (entity states corresponding to a system state).

In Figure 5, the part of the system specification produced by the Document Loan Scenario is shown. It is to be noted that a scenario transition may create several system transitions. It should also be noted that some processing is missing from this system :

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

6 Discussion/Conclusion

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

an extended finite state machine.

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

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

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

References

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

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

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

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

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

[6] 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.

[7] Ilka Kawashita. Spécification formelle des systèmes d'information par la technique des scénarios. Masters thesis, Département d'informatique et de recherche opérationnelle, Montréal, Québec, Canada, April 1997.

[8] 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.

[9] 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.

[10] 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.

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

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

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

[14] 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.

[15] François Lustman and Gilbert Babin. Formal data and behavior requirements engineering: A scenario-based approach. Research report DIUL-RR-9901, January 1999.

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