TOWARDS THE DEVELOPMENT OF
EFFECTIVE AND FLEXIBLE INFORMATION SYSTEMS

Aphrodite Tsalgatidou, Dimitris Gouscos, Constantin Halatsis
Dept. of Informatics, University of Athens, TYPA Buildings,
Panepistimiopolis, Ilisia, GR-157 71 Athens, Greece
email: {afrodite, gouscos, halatsis} @ uranus.di.uoa.ariadne-t.gr

ABSTRACT

The development of Information Systems which are effective (i.e. satisfy users' needs) and flexible enough to incorporate future changes of user requirements basically depends on the importance given to the requirements analysis stage and to the techniques employed for their development. This paper presents a new approach to requirements analysis through an environment called VENUS. The VENUS environment provides a set of tools to support the requirements capture and analysis process so that design and development is more effective and flexible. The construction of user requirements starts by first constructing an Entity-Relationship Model which is then transformed to an Object-oriented Rule-based Model (ORM) where rules are used for the definition of the behaviour of each object class. ORM is then mapped to a graphical Petri-net based model and is validated by employing graphical animation and exploiting formal properties of the underlying Petri-net formalism.

INTRODUCTION

Two desirable characteristics of Information Systems are effectiveness and flexibility. Effectiveness refers to the satisfaction of users' needs and is primarily related to the production of valid and complete requirements specifications. Flexibility refers to the ability of Information Systems to easily incorporate any future changes of user requirements; flexibility depends on the techniques employed for the development of information systems starting from the initial requirements analysis phase to the final implementation and integration stage.

The importance of requirements specifications in the information systems development process is not a recently identified need (Endres, 1975). Boehm (1975) has denoted that incorrect requirements specifications result in serious errors in the information system development process, where the cost to fix them increases as the development process progresses. However, there is still a lack of methodologies and tools which would focus on the capture and validation of requirements specifications and the work reported by Reubenstein and Waters (1991), Zave (1991) and Fickas and Finkelstein (1993) are among the few examples that can be found in this area. The large number of existing methodologies for systems analysis and design produce specifications which are inherently static. Furthermore, CASE tools play a passive rather than active role in requirements specifications. A developer, in order to validate the captured requirements, tries to derive a system's behaviour from a static description.

What is needed is an approach and a set of tools that enables a developer to obtain a 'working' model of a system at the early requirements phase, so that the system's behaviour can be better understood and appreciated by end users. This may be achieved if the produced requirements specifications are executable. An executable specification is a formal model of the system with the ability to simulate the system's behaviour when executed by a suitable interpreter; thus, such a specification can be thought of as a prototype of a proposed information system and therefore it should be comprehensible, modifiable, easy to check for internal consistency and free from bias toward particular implementation strategies.

To this end, this paper presents an integrated environment called VENUS which offers a number of tools to support the requirements engineering process. The tools offered by VENUS assist a system analyst to elicit, specify, analyse and validate executable requirements specifications within an object-oriented framework. Petri nets (Murata, 1989) are used as the underlying formalism for expressing object-oriented requirements specifications in a graphical manner. Prototyping, animation and execution are some of the mechanisms used by VENUS for validating object-oriented executable requirements specifications.

Previous studies (Tsalgatidou et al, 1994) found the object-oriented approach appropriate for developing requirements specifications in the VENUS system for a number of reasons such as:

- the uniform modelling of various real world objects
- the provision of useful abstractions, such as generalisation, aggregation and specialization, which facilitate the extension of the model and the incorporation of new types of information

- the easy construction of executable specifications.

This paper concentrates on the production of models, in particular of the Entity-Relationship Model and the Object-oriented Static Model and the validation of the produced specifications through graphical animation, and begins by presenting the VENUS environment together with its various components. The stepwise construction of requirements specifications is discussed. These specifications may firstly be expressed in an Entity-Relationship Model (ERM) which is subsequently automatically mapped to an Object-oriented Static Model (OSM). The OSM is further enhanced so as to incorporate the dynamic behaviour of the system under study and produces an Object-oriented Rule-based Model (ORM). The ORM is validated by being either translated to C++ executable specifications or mapped to a graphical Petri-net based model, called an RBN, which is then executed and graphically animated. The RBN model is formally defined and briefly introduced. Validation of the ORM by graphically animating the resultant RBN and showing that, given an initial marking of the RBN, certain markings are reachable, is also discussed. Finally, the VENUS approach is compared to related work and conclusions are drawn.

THE 'VENUS' ENVIRONMENT

The tools offered by VENUS, the interaction of the analyst with these tools, as well as the models used as input or produced by these tools are shown in fig. 1, which depicts the sequence of steps usually followed by an analyst during the requirements development process. The tools provided by VENUS are:

(a) A Graphical Editor & Text Generator for Entity-Relationship Models (ERMs), called GERM-TG. Using GERM-TG, an analyst can develop a graphical ERM which captures details about the data of the application under study. A textual description of the developed graphical ERM is automatically generated by GERM-TG (Avdis & Karpodinis, 1993).

(b) A Mapping tool, called MEROS, which takes as input ERM specifications and produces object-oriented specifications, hence constructing an Object-oriented Static Model (OSM); the OSM specifications are static since they are produced from ERM specifications which are static by definition. Thus, the OSM needs to be enhanced in order to incorporate the dynamic behaviour of the identified object classes. This is accomplished with the use of the next tool.

(c) A Syntax Directed editor, called SDORM, for enhancing OSM specifications with dynamic aspects expressed in a rule-based manner and thus producing specifications expressed in an Object-oriented Rule-based Model (ORM).

(d) A C++ generator, which takes as input ORM specifications and produces executable specifications in C++. The execution of these specifications greatly facilitates the process of validating the requirements described by an ORM.

(e) A Mapping tool, called MORBN, which takes as input ORM specifications and produces graphical RBN (Petri-net based) specifications.

(f) A Graphical Editor/Animator for the RBN model, called GEA-RBN. This tool may be used by an analyst to edit and animate graphical RBN specifications in order to validate them against certain validation criteria. This validation process is discussed in the relevant section of this paper.

Depending on the analyst's preferences and experience, the development of specifications may start at different points of the VENUS-based development process. Furthermore, the VENUS architecture allows more than one analysts to be involved in the VENUS development chain, each one working on a different phase.

THE ENTITY-RELATIONSHIP MODEL (ERM)

Many analysts are used to constructing ERMs as one of the first steps of the requirements analysis stage. Thus, although VENUS aims at producing object-oriented requirements specifications for the various reasons mentioned in the introduction, it also provides GERM-TG, thus giving to analysts the opportunity to construct a graphical ERM. Fig. 2 shows an example of a graphical ERM and its corresponding textual description. This textual description is automatically produced by GERM-TG after the graphical ERM is constructed. The BNF definition of the language for textual description of ERMs can be found in (Gouscos & Tsalgatidou, 1992).

Fig 1 The VENUS Environment

Fig. 2 An example of an ERM (Graphical & Textual Representation)

An ERM is a binary Entity-Relationship Model enriched with inheritance and constructed on the basis of a linguistic approach (Tsalgatidou & Loucopoulos, 1991). The main concepts of an ERM are: *entity types, entity subtypes, label types* and *relationship types*. Entity types describe entities with the same properties (e.g. ASSISTANT, ESSAY). Label types describe labels (i.e. values) of a certain type. Relationship types can be of two kinds: they can either associate entity types to label types, in which case they are called *reference type relationships* (e.g. COURSE has Name) or they may be confined to associating entity types, in which case they are called *fact type relationships* (e.g. ASSISTANT assists PROFESSOR). A fact type relationship may be expressed in more than one ways, showing the various *roles* of the involved entity types. For example: 'ASSISTANT assists PROFESSOR' and 'PROFESSOR assisted_by ASSISTANT', where 'assists' and 'assisted_by' are the roles of the entity types ASSISTANT and PROFESSOR respectively in this relationship. Cardinality constraints are also modelled by ERM, for example 'ASSISTANT assists 1:1 PROFESSOR' and 'PROFESSOR assisted_by 1:N ASSISTANT'.

THE OBJECT-ORIENTED MODEL

The Object-oriented Static Model (OSM)

OSM is an object-oriented model (Gouscos, 1992) which encapsulates only the static aspects of requirements and is automatically derived from ERM using the MEROS tool. Fig. 3 contains part of the OSM specifications which were generated by applying the MEROS tool to the ERM of fig. 2. It can be seen that entity types have been mapped to object classes, whereas relationship types have been mapped to properties of the corresponding object classes.

```
object schema ESSAYS
    value classes
        ESSAYSTATUS values [submitted, under_init_review, ..., for_revision]
        .............
    end value classes
    object classes
        .............
        object class PROFESSOR subclass of PERSON
            instance properties
                Teaches 1:1 COURSE
                AssistedBy 1:N ASSISTANT
            end instance properties
        end object class PROFESSOR
        .............
    end object classes
end object schema ESSAYS
```

Fig. 3 An example of an OSM specification

This is a first-cut object-oriented model which must be subsequently enchanced by a systems analyst so as to incorporate the dynamic aspects of the identified object classes (reflecting the business policy) and any new object classes that may be needed. The SDORM tool provided by VENUS is a syntax-directed editor which assists the analyst in this enhancement of OSM. The output of this process is an Object-oriented Rule-based Model (ORM) which is briefly described in the following.

The Object-oriented Rule-based Model (ORM) ∖

ORM is an extension of OSM incorporating the dynamic aspects of the identified object classes using the concepts of *behaviour units (BUs), signals* and *rules*. The concept of BUs is used to partition the behaviour of an object class. Each BU belongs to one class of the object-oriented model and is associated with a specific triggering signal type. The receipt of individual signals of this type activates the behaviour described in this BU. A BU may be either class BU or instance BU and has the following general structure:

```
class | instance BU class_name.BU_name triggered by signal_name(signal_parameters)
    BU body
end BU BU_name
```

The body of a BU is a set of dynamic rules having the form:

[**IF** preconditions **THEN**] actions

The preconditions of a rule are expressed by a boolean formula and have to be satisfied before the actions described in the rule's *THEN part* can be executed. The receipt of a triggering signal by a BU activates all the dynamic rules of that BU. The preconditions of rules of the same BU are mutually exclusive, so that exactly one of the rules will always fire. A rule with no preconditions executes its actions every time an appropriate triggering signal is received. Actions in a rule may *modify/create/delete* object instances and/or *produce* some *signals* sent to other BUs or to the external environment of the modelled system. A rule cannot be part of more than one BU and a BU belongs to exactly one class; therefore the dynamic behaviour of each object class is modelled as a collection of rules grouped in the BUs specified for that class and triggered by specific signals.

```
object class PROFESSOR subclass of PERSON
    instance properties
        Teaches 1:1 COURSE
        AssistedBy 1:N ASSISTANT
    end instance properties
    .....................
    instance behaviour
        instance BU PROFESSOR.DecideForEssay triggered by DecideForEssay (Essay,Review)
            rule R1 hac PROFESSOR lac {PROFESSOR}
            If signal.Review.Mark in {Aplus, A, B} then
                produce signal SetStatus(accepted) for signal.Essay;
                produce signal NewPass(signal.Essay.WrittenBy, signal.Essay.WrittenFor, signal.Review.Mark) for
                    PASS;
                produce signal EssayAccepted(signal.Essay, signal.Review) for signal.Essay.WrittenBy;
            rule R2 hac PROFESSOR lac {PROFESSOR}
            If signal.Review.Mark in {D,E} then
                produce signal SetStatus(rejected) for signal.Essay;
                produce signal EssayRejected(signal.Essay, signal.Review) for signal.Essay.WrittenBy;
            rule R3 hac PROFESSOR lac {PROFESSOR}
            If signal.Review.Mark = C then
                execute DefineDeadline(signal.Essay, signal.Review,Deadline);
                produce signal SetStatus(waiting_for_revision) for signal.Essay;
                produce signal SetDeadline(Deadline) for signal.Review;
                produce signal EssayForRevision(signal.Essay, signal.Review) for signal.Essay.WrittenBy;
        end BU DecideForEssay
        ................
    end instance behaviour
    ..................
end object class PROFESSOR
```

Fig. 4 An ORM specification

ORM supports single inheritance as follows: object classes inherit all the static properties of their superclasses and may introduce new ones. They also inherit the BUs of their superclasses without any changes and/or redefine (some of) them by changing (some of) their existing rules or by adding new ones. Object classes may also introduce new BUs describing behaviour not specified by their superclasses. For each rule R, the rule model provides two fields: *highest acceptor class (hac(R))* containing the name of the class where rule R is defined and *lowest acceptor classes (lac(R))* containing the names of the lowest subclasses of hac(R) that still inherit rule R.

Fig. 4 contains part of the ORM that resulted from the enhancement of the OSM of fig. 3. More specifically, fig. 4 depicts the part of object class PROFESSOR which specifies what an instance of this class has to do in order to decide for an Essay that was Reviewed. This is described in terms of a BU (called PROFESSOR.DecideForEssay) triggered by a signal (called DecideForEssay(Essay,Review)). The body of this BU is a group of three rules which, depending on the satisfaction of certain preconditions, execute appropriate actions. The *hac* and *lac* fields of all three rules in this example are set to PROFESSOR and {PROFESSOR} respectively, meaning that these rules have been defined in this class and apply only in this class, i.e. they are not inherited by any other classes. The rules which have been inherited by class PROFESSOR from class PERSON (not

shown in this figure) have their *hac* and *lac* fields set to PERSON and {PROFESSOR} respectively. In case that a rule of class PERSON is also inherited by classes ASSISTANT and STUDENT, then its *hac* field is again set to PERSON, whereas its *lac* field is set to {PROFESSOR, ASSISTANT, STUDENT}.

Therefore, the dynamic behaviour of an information system is modelled as a sequence of firings of rules, grouped in BUs and triggered by specific signals. All the rules of a BU are triggered by the same signal. The firing of each rule, may result in the activation of other BUs by generating appropriate signals. The interaction of an information system with its external environment is modelled by the receipt and sending of signals by BUs. Thus, signals serve as the means of communication between the various parts of an information system and between the system and its external environment. More about ORM may be found in (Tsalgatidou et al, 1993).

The dynamic behaviour of an ORM can be validated by executing the resultant specifications. VENUS offers a C++ generator tool which is used for transforming ORM specifications to C++ executable specifications (Tsalgatidou et al, 1994). Furthermore, VENUS offers a tool called MORBN, which produces a graphical Petri-net representation of an ORM, called RBN. RBN is a formal, graphical and executable model. The dynamic behaviour of a RBN can be demonstrated by employing the graphical animation and execution facilities provided by the GEA-RBN tool of VENUS. Furthermore, system behaviour can be observed at a number of abstraction levels. The mapping of ORM to RBN and the formal definition of the latter are presented in the following section.

THE RULE-BASED NET (RBN) MODEL

A Rule-Based Net is a Petri net-based model which is used for a graphical modelling of the ORM. An RBN resembles a Predicate-Transition (PrT) net (Genrirch & Lautenbach, 1981), augmented with additional information. The basic constituents of an RBN are places, transitions and arcs connecting places and transitions. Each place is inscribed with a signal type and may hold tokens which represent signals of this signal type. Each transition has a unique input place and is inscribed with rules which are triggered when the corresponding input place holds a token, thus denoting that a signal is present. When the *IF part* of a triggered rule is satisfied, the rule executes its actions. Each arrow is inscribed with the number and type of signals that can flow through it in order to activate a rule or to collect signals into a place. An RBN also contains an underlying textual structure which describes information used in the various inscriptions.

An analyst may interactively produce RBNs at different abstraction levels using the MORBN tool. Five distinct abstraction levels may be produced:

1st abstraction level -> 'context RBN';
>   this RBN contains only one transition corresponding to the behaviour of the entire modelled system; places represent signals sent to and from the external environment.

2nd abstraction level -> 'high-level RBN';
>   here, the unique transition of the 'context RBN' is decomposed in other transitions and places corresponding respectively to the object classes of ORM and to the signals exchanged between these classes.

3rd abstraction level -> 'medium-level RBN';
>   each 'class transition' of the 'high-level RBN' is further decomposed here into other transitions and places, corresponding respectively to the BUs of this object class and to the signals exchanged between these BUs.

4th abstraction level -> 'low-level RBN';
>   in this RBN, each 'BU transition' is decomposed into transitions corresponding to the rules contained in this BU. These new transitions are inscribed with the corresponding rule names.

5th abstraction level -> 'detailed RBN';
>   this RBN does not contain more transitions or places than the 'low-level RBN'; the difference is in the inscription of transitions which now contain the 'body' of rules (namely the *hac, lac, preconditions* and *actions* of each rule). This 'detailed RBN' is always accompanied by its structure S.

A RBN of the lowest abstraction level, i.e. a 'detailed RBN' is formally defined as a tuple

$$R = < P, T, F, K, N, \_ , sig, insc, hac, lac, M_0 >$$

where **P**, **T** and **F** are sets corresponding respectively to RBN places, transitions and arcs; **K** and **N** are functions which respectively map each place and each arc to an integer number denoting the capacity of the place or the multiplicity of the arc; _ is the underlying structure of an RBN which denotes the domain of signal parameters, of the properties of object instances, of objects of the RBN and the signal domain of the RBN; **sig, insc, hac** and **lac** are functions which relate the various RBN elements to appropriate elements of ORM and finally M_0 is an initial marking that assigns token to places.

A detailed definition of an RBN as well as examples for RBNs of various levels may be found in (Tsalgatidou et al, 1994). Fig. 5 depicts an example of a 'low-level RBN', i.e. an RBN where transitions are inscribed with rule names and places are inscribed with appropriate signals which are produced by and/or trigger rules.
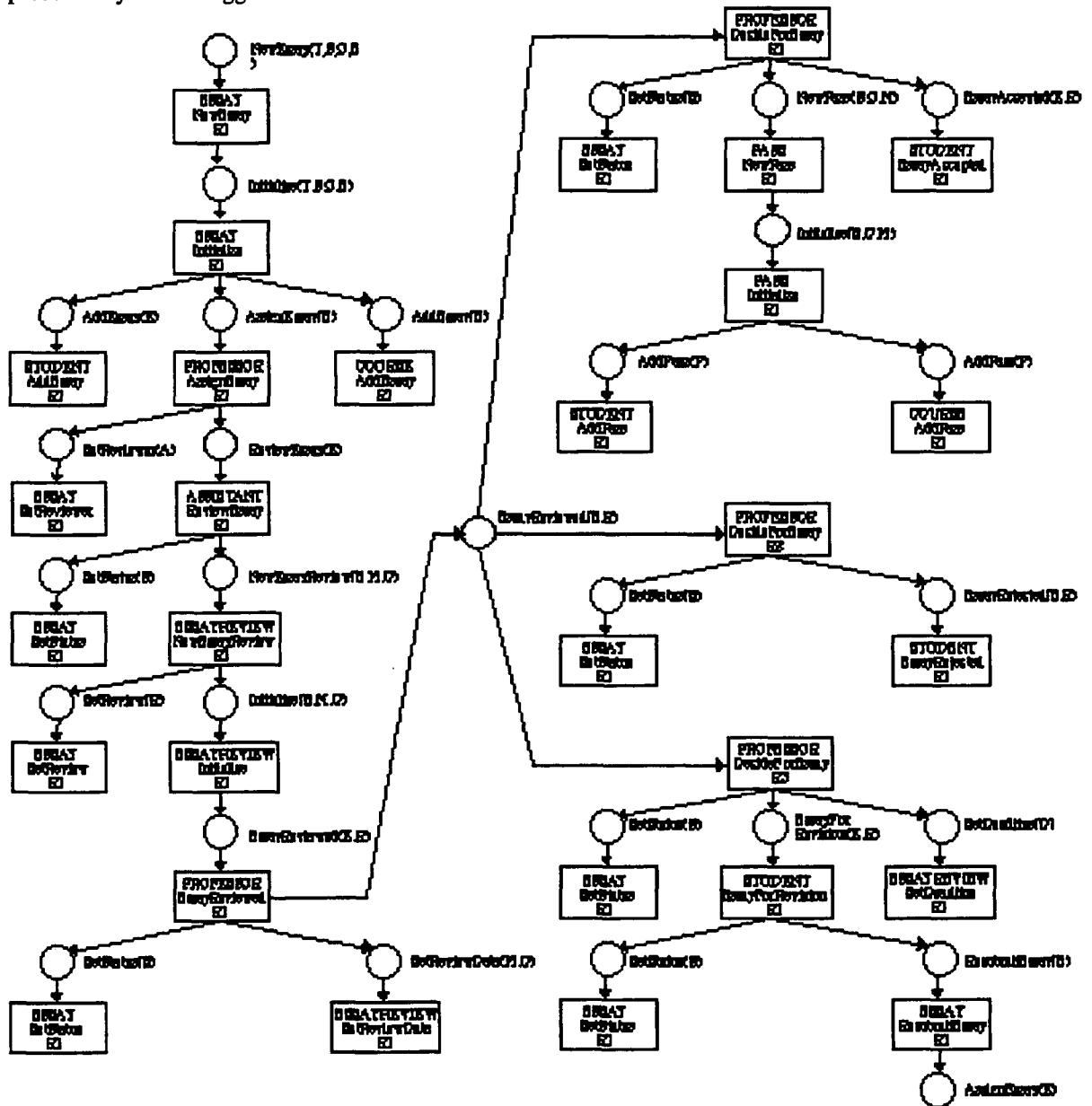

Fig. 5 A 'low-level' RBN

Thus, an RBN is a simple formal model high-lighting *control flow* within a system and hiding at the same time, in the corresponding transitions, information concerning *what* is happening in the system. One of the advantages of using RBNs for modelling the dynamic behaviour of information systems is that the resulting graphical representation facilitates validation, as well as analytical study of system behaviour by exploiting certain properties of the nets; this topic is discussed in the next section.

VALIDATION OF REQUIREMENTS SPECIFICATIONS

Validation of user requirements is a hard process. It refers to the production of specifications which are complete and consistent without ambiguities and vagueness. One way of validating the ORM requirements is by executing the generated C++ prototype, as has already been mentioned. Another way of ORM validation is through the constructed RBN model due to the executable nature and the potential for graphical animation of the latter. Animation has proven a valuable validation tool, and the results of early animation are very promising way for reducing errors during requirements specifications and ensuring that the intended system behaviour has indeed been properly captured and modelled.

Fig. 6 is an example of validation through graphical animation of a 'high-level' RBN, i.e. an RBN where transitions correspond to object classes of the modelled application and places represent the signals exchanged between object classes. In this example, a student creates a new essay and this is shown by a token in the signal place called NewEssay (T,S,C,B) (see fig. 6(a)). This signal is directed to an appropriate instance of class ESSAY, triggers the corresponding BU and the rule of this BU which has its IF part satisfied will fire, i.e. execute its actions. In the next snapshot (6(b)), there are two tokens assigned to places SetReviewer(A) and ReviewEssay(E); this means that the appropriate rule of class ESSAY has already fired, and a signal of type AssignEssay(E) has been generated and sent to an instance of class PROFESSOR triggering the appropriate BU. Subsequently, an appropriate rule of the triggered BU has fired and has generated two signals of type SetReviewer(A) and ReviewEssay(E) which are represented by the two tokens shown in fig. 6(b).The signal of the SetReviewer(A) type is directed to the appropriate instance of class ESSAY whereas the signal of the ReviewEssay(E) type is directed to the appropriate instance of class ASSISTANT and triggers the rules that deal with the process of reviewing an essay by an assistant.

The snapshot of fig. 6(c) shows two tokens at the places SetReview(R) and EssayReviewed (E,R) which are output places for the ESSAYREVIEW class. This means that the appropriate assistant has reviewed the essay and sent a signal of type NewEssay Review(E,M,C) to the object class ESSAYREVIEW in order to activate the rule which generates a new instance of this class and subsequently produces the two signals shown in this snapshot (fig. 6(c)). At the last snapshot (fig. 6(d)), depicts three tokens, in places EssayRejected(E,R), SetStatus(S) and SetReviewData(M,C) respectively. This marking means that a professor, taking into account the comments and the marks assigned to the essay by his/her assistant, decided to reject the essay. Thus, a signal of type EssayRejected(E,R) is sent back to the student in order to inform him/her about the results of essay revision; furthermore, the appropriate instances of classes ESSAY and ESSAYREVIEW receive a signal of type SetStatus(S) and SetReviewData(M,C), respectively, in order to update the information held about the status of the essay and its review data. This is the final marking of the review process of an essay submitted by a student.
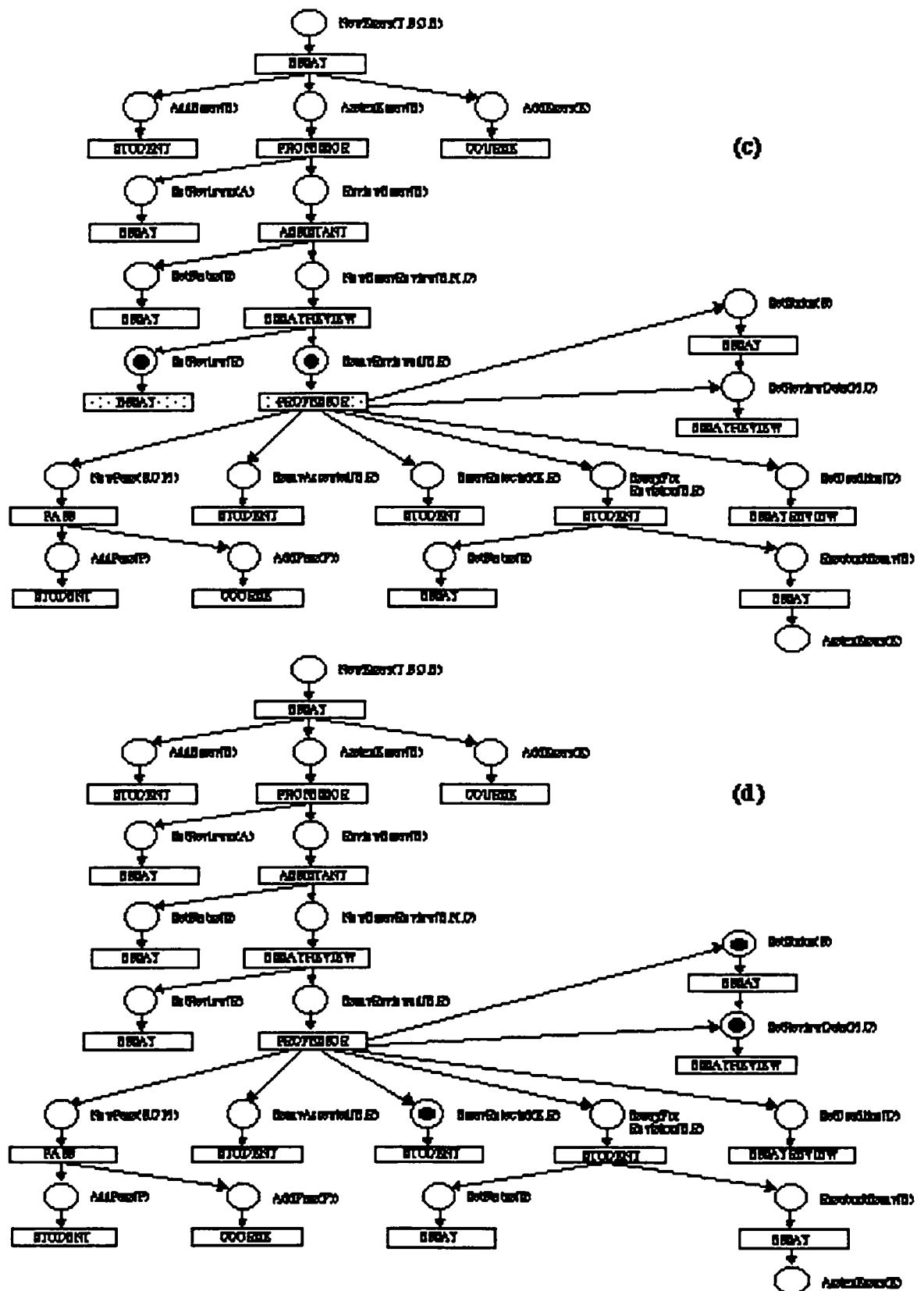
Fig. 6 (a,b) (shaded transitions fire to produce next marking)

Fig. 6 (c,d) (shaded transitions fire to produce next marking)

The Graphical Editor and Animator for RBNs (GEA-RBN) (currently under development on Unix workstations using OSF/Morif, C++ & Prolog) provided by the VENUS environment, enables animation in two ways: a) user-driven (guided by an analyst working together with end-users of the modelled application) and b) automatic. In the former case, the user decides which rules will fire and which will not, and comments on results. In the latter case, GEA-RBN is given an initial signal to

evaluate all alternative paths (i.e. preconditions of all triggered rules), execute the corresponding actions and produce various alternative markings. Thus, it is checked whether various markings are reachable, i.e. whether all prescribed states can be reached and under which conditions. Furthermore, missing rules, dead-end rules (i.e. rules which can never fire), wrong rules (i.e. rules producing wrong results) or circular rules can also be identified. Animation feedback can be used to 'weed-out' any errors discovered, thus helping in the interactive design of the rule base.

Therefore, using GEA-RBN one can prove that the produced RBN specification is valid, i.e. that it is *deterministic, complete, consistent* and *correct*. An RBN specification is considered *deterministic* when the behaviour of the modelled system is always predictable; this follows from the premise that information systems are considered to be deterministic, i.e., certain things happen under certain conditions. This means that whenever a signal arrives from the external environment or is generated by the system, we always know what happens next. The RBN specification is *complete*, when there is always something happening, whenever a signal arrives. The rule-based specification is *consistent*, when there are no rules with the same triggering signal and the same preconditions but with contradictory actions. Finally, the resulting requirements specifications are *correct*, when they do what end-users want their system to do.

RELATED WORK

Requirements specifications are produced in the VENUS environment within a rule-based object-oriented framework. There are many examples of the use of the object-oriented approach in systems analysis and development in Van Baelen et al (1992), Booch (1991), Coad & Yourdon (1991), Coleman et al (1992), Rumbaugh et al (1991). Furthermore, examples of the use of rules in requirements specifications may be found in D'Haenens et al (1991) and Loucopoulos et al (1991). The VENUS environment combines both the object-oriented and rule-based approaches within a uniform modelling framework for all system aspects.

VENUS also places a lot of emphasis on the production of executable specifications. The importance of executable specifications has been realised by researchers and developers within the software engineering community and a number of executable system models, executable specification languages and tools contributing in this direction have appeared lately. For example PAISLey (Zave, 1991) is a language designed for real-time distributed systems which uses an operational approach for building a requirements specification, emphasizing the definition of processes as the building blocks of the system. VENUS differs from PAISLey in the sense that the requirements specifications are object-oriented and the basic building blocks of the system are objects rather than processes.

Graphical animation and Petri nets are among the mechanisms and tools used by VENUS for modelling and validation requirements specifications. Animation, on the one hand, has proven to be a valuable validation technique; see, for example, the work reported by Finkelstein and Kramer (1992), Dahler and his colleagues (1987) and Shand et al (1988). On the other hand, Petri nets are a powerful formalism (Murata, 1989) and have been used as a modelling tool in various applications. For example, in Lakos & Keen (1993) a language for object-oriented Petri nets (LOOPN) is used for modelling a door controller protocol. Other examples may be found in Brinkkemper & der Hofstede (1990) and Kappel & Schrefl (1991), where extensions of Petri nets are used for bridging the gap between informal requirements engineering activities and the more formal system development activities. VENUS introduces a rule-based extension of Petri nets for modelling and animating object-oriented requirements specifications for validation purposes.

Therefore, the VENUS approach is related to object-oriented development, produces executable specifications and uses Petri nets as the underlying formalism and graphical animation for modelling and validating requirements specifications. The major advantage of VENUS over other approaches is the integration of a number of paradigms into a uniform modelling framework.

SUMMARY AND CONCLUSIONS

The development of effective and flexible information system starts from the production of complete and valid requirements specifications. This paper presented the VENUS uniform framework which demonstrates the combination of object-oriented and rule-based approaches, integrating all information system aspects. Requirements specifications produced by VENUS preserve explicit representation of application domain policy in terms of dynamic rules. These object-oriented rule-

based specifications are mapped to the formal and graphical RBN model for validation purposes and this mapping is well defined and automated.

Validation of RBNs is performed through graphical animation, offering the advantages of rapid prototyping. Thus, at early development stages of a system, an analyst can produce easily modifiable, modular, clear, concise, compact and executable requirements specifications. Two other interesting features are the modelling of the dynamic behaviour of a system at different levels of abstraction and formality and the modelling of the behaviour of one object class in one transition of a 'high-level RBN'. This latter feature complies to the localisation principle, i.e. the description of objects in isolation from others, see Rolland (1992), since all the rules and signals related to one object are localised in the corresponding 'high-level RBN' transition, instead of being spread out in a huge RBN for the whole system under study, as it is the case in classical behaviour models.

The current version of VENUS already demonstrates a number of capabilities that can be used to bridge the gap between informal and formal specifications. The analyst is responsible for communicating with the end-user and entering the required information in the system. VENUS supports the analyst by offering tools (graphical and syntax-directed editors and text generators, C++ generator, RBN graphical editor/animator) for constructing and validating requirements specifications. VENUS provides specific support for evolutionary construction of requirements, using the facilities of the underlying system to ensure that each change is carried out in a consistent way thoughout the requirements specification process.

We believe that the work described in this paper contributes to the development of systems that satisfy the initial user requirements and that are flexible enough to incorporate and reflect any future changes. Future work will focus on the mapping of the produced executable requirements specifications to design and implementation structures. Furthermore, a full-scale prototype will be constructed and tested on full-scale requirements and on the development of algorithms for mapping RBN specifications to design and implementation structures.

REFERENCES

Avdis, T. & Karpodinis, K. (1993). *The ERM and its graphical editor and text generation tool*. B.Sc. Dissertation, November, Dept. of Informatics, University of Athens.

Van Baelen, S., Lewis, J., Steegmans, E., Van Riel, H. (1992). "EROOS: An Entity-Relationship Based OO Specification Method", *Proceedings of the 7th International Conference TOOLS EUROPE 1992*, Prentice-Hall Int. Ltd, pp. 103-118.

Boehm, B. W. et al. (1975). Some experiences with automated aids to the design of large-scale reliable software, *IEEE Transactions on Software Engineering*, Vol. 1, No. 2.

Booch, G. (1991) *Object-Oriented Design with Applications*, Benjamin/Cummings Publ. Co., Inc.

Brinkkemper, S. & der Hofstede, A.H.M. (1990) "The Conceptual Task Model: a Specification Technique between Requirements Engineering and Program Development", *Proceedings of CAiSE 1990*, Lecture Notes in Computer Science, vol.436, Springer-Verlag, pp. 228-250.

Coad, P. & Yourdon, E. (1991) *Object-Oriented Analysis*, Prentice-Hall Int.

Coleman, D., Hayes, F. & Bear, S. (1992) "Introducing Objectcharts or How to Use Objectcharts in Object-Oriented Design", *IEEE Transactions on Software Engineering*, January, Vol. 18, No.1, pp. 9-18.

D'Haenens, I., Van Assche, F., Halpin, E. & Karakostas, B. (1991) "Experiences with Rule-Based Dynamic Modelling", in (Sol & van Hee, 1991).

Dahler, J. et al. (1987) "A Graphical Tool for the design and prototyping of distributed systems", *ACM SIGSOFT*, Vol.12, No.3, pp. 25-36.

Finkelstein, A & Kramer, J. (1992) "TARA: Tool-Assisted Requirements Analysis", in (Loucopoulos & Zicari, 1992), pp. 413-432.

Endres, A. (1975) "An analysis of errors and their causes in system programs", *IEEE Transactions on Software Engineering*, June, Vol. 1, No. 6, pp. 140-149.

Fickas, S & Finkelstein, A. (eds.) (1993) *Proceedings of IEEE International Symposium on Requirements Engineering*, San Diego, CA, 4-6 Jan., IEEE Computer Soc. Press.

Genrich, H. J. & K. Lautenbach, K. (1981) "System Modelling with High-Level Petri Nets", *Theoretical Computer Science*, Vol. 13, pp. 109-136.

Gouscos, D. & Tsalgatidou, A. (1992) "The ERM, ORM and RBN Models", Technical Report, May, Dept. of Informatics, University of Athens.

Kappel, G. & Schrefl, M. (1991) "Using an Object-Oriented Diagram Technique for the Design of Information Systems", in (Sol & van Hee, 1991).

Lakos, C.A. & Keen, C.D. (1993) "Modelling a Door Controller Protocol in LOOPN, *Proceedings of the 10th International Conference TOOLS EUROPE 1993*, Versailles, France, Prentice-Hall, pp. 31-44.

Loucopoulos, P., McBrien, P., Schumacker, F., Theodoulidis, C., Kopanas, V. & Wangler, B. (1991). "Integrating Database Technology, Rule-Based Systems and Temporal Reasoning for Effective Information Systems: The TEMPORA Paradigm", *Information Systems*, vol. 1, no. 1.

Loucopoulos, P. & Zicari, R. (eds.) (1992) *Conceptual Modeling, Databases, and CASE*, John Wiley & Sons, Inc.

Murata, T. (1989) "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, April, Vol. 77, No. 4.

Reubenstein, H. B. & Waters, R. C. (1991) "The Requirements Apprentice: Automated Assistance for Requirements Acquisition", *IEEE Transactions on Software Engineering*, March, Vol. 17, No.3, pp. 226-240.

Rolland, C. (1992). "Trends and Perspectives in Conceptual Modelling", in (Loucopoulos & Zicari, 1992), pp. 27-48.

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

Shand, J et al. (1988) "An enviroment for the execution and graphical animation of JSD specifications", in *Procs. of International Workshop of KBS in S/W Engineering*, UMIST, Manchester.

Sol, H.G. & van Hee, K.M. (eds.) (1991) *Dynamic Modelling*, North-Holland.

Tsalgatidou, A. & Loucopoulos, P. (1991) "An Object-Oriented Rule-Based Approach to the Dynamic Modelling of Information Systems", in (Sol & van Hee, 1991).

Tsalgatidou, A., Gouscos, D. & Halatsis, C. (1993) "Rule-Based Behaviour Modelling of Information Systems", *Proceedings of the 26th Hawaiian Conference on Systems Sciences (HICSS-26)*, Vol. IV, January, IEEE Computer Society Press, pp. 409-418.

Tsalgatidou, A., Gouscos, D. & Halatsis, C. (1994) "Specifying and Validating Requirements: The VENUS system", to appear in the *Proceedings of the 11th International Conference on Technology of Object-Oriented Languages and Systems - TOOLS EUROPE 1994*, March, Paris, Prentice-Hall, Int. Ltd., pp. 89-102.

Zave, P. (1991) "An Insider's Evaluation of PAISLey" *IEEE Transactions on Software Engineering*, Vol. 17, No. 3, pp. 212-225.