Animation of Conceptual Models using two Concurrent Environments: an overview*

Patricio Letelier Pedro Sánchez Isidro Ramos
Department of Information Systems and Computation
Valencia University of Technology, 46020 Valencia
SPAIN
{letelier, ppalma, iramos}@dsic.upv.es
http://www.dsic.upv.es/users/oom

Rafael Corchuelo
Department of Languages and Information Systems
University of Sevilla, 41012 Sevilla
SPAIN
corchu@lsi.us.es

Abstract

Abstract: $\mathcal{OASIS}$ is a formal approach for the specification of object oriented conceptual models. In $\mathcal{OASIS}$ conceptual schemas of information systems are represented as societies of interacting concurrent objects. Animating such models in order to validate the specification of information systems is a topic of interest in requirements engineering. Concurrent Logic Programming and Petri nets are suitable models for distributed computation allowing a natural representation of concurrence. Using Concurrent Logic Programming or Petri Nets, $\mathcal{OASIS}$ specifications are animated according to $\mathcal{OASIS}$ execution model. In this paper, we show our work in translating $\mathcal{OASIS}$ concepts into concurrent environments. This work has been developed in the context of a CASE tool supporting the $\mathcal{OASIS}$ approach. Our aim is to build a module for animation and validation of specifications. A preliminary version of this module is presented.

Key-Words: Formal Specification, Animation of Specifications, Object-Oriented Models.

1 Introduction

Conceptual models, representing the functional requirements of information systems, are a key factor when linking the problem and solution domains. Building a conceptual model is a discovery process, not only for the analyst but also for the stakeholders. The most suitable strategy in this situation is to build the conceptual model in an iterative and incremental way, through analyst and stakeholder interaction. Conceptual modeling involves four activities: elicitation of requirements, modeling or specification, verification of quality and consistency, and eventually, validation.

Formal methods for conceptual modeling provide improvements in soundness and precision for specifications, simplifying their verification. However, when considering elicitation and requirements validation, prototyping techniques are more used. Hence, it is interesting to obtain a combination of both approaches.

This work uses $\mathcal{OASIS}$ [8] (**O**pen and **A**ctive **S**pecification of **I**nformation **S**ystems) as a formal approach for object-oriented conceptual specification of information systems. This is a step forward in a growing research field where validation

*This research is supported by the "Comisión Interministerial de Ciencia y Tecnología" (CICYT) through the MENHIR proyect (grant no. TIC97-0593-C05-01).

of formal specifications through animation is being explored [14]. In this sense, some other proposals close in nature to $\mathcal{OASIS}$ are [6] and [5]. The differences, though, between these works and ours are basically determined by features of the underlying formalisms and the offered expressiveness. According to the presented results, the state of art is similar and is characterized by preliminary versions of animation environments.

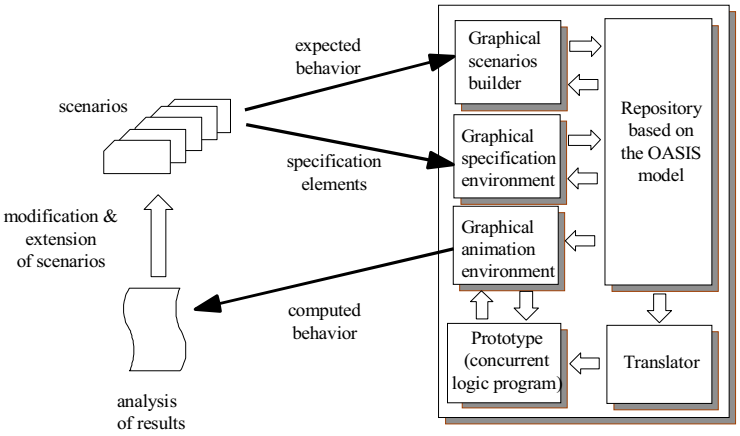

Figure 1: A framework for incremental specification of requirements

Fig.1 shows our framework for elicitation, modeling, verification and validation of requirements. Elicitation is achieved by using scenarios [12]. The expected behavior and elements of a given specification are extracted by the analyst from scenarios. The graphical scenario builder helps define scenarios in a suitable way. Functional requirements are modeled using a graphical specification module based on $\mathcal{OASIS}$. Conceptual models can be verified according to $\mathcal{OASIS}$ formal properties. At each stage of the requirements specification process it would be possible to validate the behavior of the associated prototype against the expected behavior. This comparison could lead to updates or extensions of existing scenarios. This cycle continues until the requirements are compliant with the proposed set of scenarios.

Experiments have been carried out using Object-Oriented Petri Nets [13] and Concurrent Logic Programming [9] as semantic domains for $\mathcal{OASIS}$ specifications. Correspondences between $\mathcal{OASIS}$ and these environments have been implemented in a translator program. The translator takes an $\mathcal{OASIS}$ specification stored in the repository and generates automatically a Concurrent Logic Program or a Petri Net that constitutes a prototype for the corresponding conceptual model. Furthermore, through a preliminary version of the graphical animation environment, the analyst can interact with the prototype in a suitable way. Regarding Concurrent Logic Programming, we have worked with the concurrent logic languages Parlog [2] and KL1 [1]. Regarding Petri Nets we have worked with Object-Oriented Timed Petri Nets [3] and Object Petri Nets [7].

2 Why a concurrence model?

Considering the utility of this work in requirements validation of $\mathcal{OASIS}$ specifications, it is essential to obtain the nearest implementation solution to the $\mathcal{OASIS}$ semantics. If so, we will have a way to provide formalization for the translation process. In this way, we think the best choice is to give each object an execution thread.

Fig.2 shows four ways of implementing concurrent object systems. The horizontal axis goes from purely sequential to purely concurrent. The vertical axis goes from a no object-oriented paradigm to a completely one. In three of these ways, a *monitor* should be necessary to control the activity of the system. The first (sequential, -OO) is

the simplest: there is only one monitor and there are objects whose state can be represented by a relational database, for instance. The second (concurrent, -OO) gains in concurrence by increasing the number of active processes. However there is no object representation. The third way considers each object as an independent unit of execution but there is a quantum of running time for each object. The fourth, which we chose in our model, is fully concurrent and each object is considered as an independent unit of execution. Concurrent Logic Programming and Petri Nets allow representing directly systems of this kind.

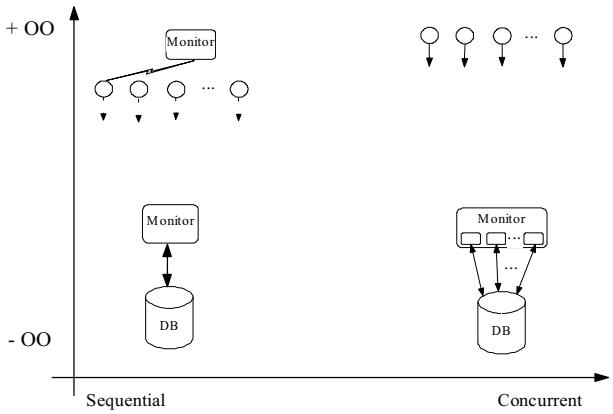

Figure 2: Implementation of concurrent object systems.

3 $\mathcal{OASIS}$

$\mathcal{OASIS}$ is a formal object-oriented language for the specification of information systems at a high level of abstraction. An $\mathcal{OASIS}$ specification is a presentation of a theory in the used formal system and is expressed as a structured set of class definitions. Classes can be simple or complex. A complex class is defined in terms of other classes (simple or complex). Complex classes are defined by establishing relationships among classes. These relationships express aggregation or inheritance. A class has a name, one or more identification mechanisms for its instances (objects) and a type or template that is shared by every instance belonging to the class. The state of the object in a given instant will depend on the actions occurred in its lifetime until the moment during which it is observed. In this way, an object is seen as an observable process. Thus, an object has a life characterized by the occurrence of actions (that could be either requested or provided). In this way, an object is seen as client or server depending on whether the object is requesting or providing services.

Services provided in an atomic level by an object are referred as events. Each object has a creation event (starting its life) and optionally a destruction event (finishing its life). Events can be organized in a molecular level as processes. Beside the own semantics used when specifying processes, we add another one in order to make a distinction between prohibition (**protocol**) and obligation process (**operation**). An obligation process is a service offered by an object in a higher level. Particularly, a transaction is an obligation process having the implicit all or nothing policy. A prohibition process does not allow the execution of some sequences of actions in an object life. In $\mathcal{OASIS}$, an information system is regarded as a society of autonomous and concurrent objects interacting with each other by the occurrence of actions. An action is a tuple including the client, the server and the requested service.

The visibility between objects is determined by a interfacing mechanism. Every object encapsulates its own state and behavior rules. As usual in object-oriented environments, objects can be seen from two points of view: static and dynamic. From the static perspective, the attributes are the set of properties describing the object structure.

The object state in a definite instant is the set of structural properties values. From the dynamic perspective, the evolution of objects is characterized by the "change of state" notion. The occurrence of actions implies changes (by means of **valuations** and **derivations**) in the values of the attributes. Object activity is determined by a set of rules: **preconditions**, **integrity constraints**, **triggers**, **protocols** and **operations**. A step is the set of actions executed at the same instant by the object. Every object has an unique identifier (oid) set by the system. Objects are referred by one or more identification mechanisms belonging to the problem space. An identification function sets a mapping between the identification mechanisms and the oid. The type or template describes the structure and behavior of every object.

$\mathcal{OASIS}$ is a formal specification language that allow defining conceptual schemas according to the object model that has been briefly presented.

3.1 $\mathcal{OASIS}$ semantics

The semantics of $\mathcal{OASIS}$ is defined by means of *Kripke* structures (W, τ, ρ) which include an universe of states w. To each class we associate an accessibility relation ρ in such a way that a pair of states (s, t) is in that relation if and only if there is a transition going from the state s to the state t. Let us call W the set of all possible worlds reachable by an object. Let us call F the set of state well formed formulae (wff) being evaluated over the current state of the object. Let A be the set of ground actions of the object template and $2^{\underline{A}}$ the set of all possible instantiated steps. The functions τ and ρ are defined as follows:

$$\tau : F \rightarrow 2^W$$
$$\rho : 2^{\underline{A}} \rightarrow (W \rightarrow W)$$

The function τ says in which worlds a state formula (in First Order Predicate Logic) is true. The function ρ is a binary relation between worlds (declarative semantics of the language). Given a step $\mu \in 2^{\underline{A}}$, $w, w' \in W$ then $(w, w') \in \rho(\mu)$ if and only if occurring μ implies the transition between the worlds w and w'.

3.2 $\mathcal{OASIS}$ expressed in DL

In [10] Deontic Logic is described as a variant of Dynamic Logic (DL) [4]. The definition of Deontic operators in Dynamic Logic is:

$\psi \rightarrow [a]false$	"the occurrence of a is forbidden in states where ψ is satisfied".
$\psi \rightarrow [\neg a]false$	"the occurrence of a is obligatory in states where ψ is satisfied".
$\psi \rightarrow [a]\phi$	"in states where ψ is satisfied, immediately after a occurrence, ϕ must be satisfied"

where ψ is a well formed formulae that characterizes an object state when the action a occurs and $\neg a$ represents the non-occurrence of the action a (i.e., only other actions different from a could occur). Furthermore, there is no state satisfying the atom $false$. This represents a state of system violation. Thus, one action is forbidden if its occurrence leads the system towards a violation state and one action is obligatory if its non-occurrence leads the system towards a violation state. The $\mathcal{OASIS}$ template is mapped to the formulae previously presented.

These formulae constitute a sublanguage of the language proposed and formalized in [15]. In [8] $\mathcal{OASIS}$ is presented as a specification language with a well defined syntax. Here is an example of part of a simple vending machine system using the $\mathcal{OASIS}$ syntax.

```
conceptual schema vending_machine_system
class vending_machine
identification
    number: (number);
constant attributes
    number: nat.
variable attributes
    n_chocs: nat;
    credit: nat(0);
    switch_on: bool(false);
events
    set new;
    coin_in;
    coin_out;
    choc;
    light_empty;
operations
    CANCEL:
      CANCEL1= {credit>1}::coin_out.CANCEL1
               + {credit<=1}::coin_out;
triggers
    ::light_empty when
        {nchocs= 0 and switch_on= false};
valuations
    [coin_in] credit=credit+1;
```

```
    [::coin_out] credit=credit-1;
    [choc] nchocs=nchocs-1, credit=credit-1;
    [::light_empty] switch_on=true.
preconditions
    choc if {credit>0 and nchocs>0};
    CANCEL if {credit>0};
protocols
    GETCHOC:
        GETCHOC1= coin_in.GETCHOC2;
        GETCHOC2= choc.GETCHOC1 +
                  coin_in.GETCHOC3 +
                  ::coin_out.GETCHOC1;
        GETCHOC3= choc.GETCHOC2 +
                  coin_in.GETCHOC4 +
                  ::coin_out.GETCHOC2;
        GETCHOC4= choc.GETCHOC3 +
                  ::coin_out.GETCHOC3;
end class

class customer
identification
    name:(name);
constant attributes
    name:string;
events
    add new;
    remove destroy;
end class


interface customer(someone)
        with vending_machine (someone)
    services(coin_in,choc);
end interface


interface vending_machine(someone) with self
    services(light_empty, coin_out);
end interface
end conceptual schema
```

In this example there are two classes: `customer` and `vending_machine`. The objects in both classes are active. A `vending_machine` object is forced to self-trigger an action with the event `light_empty` whenever the trigger condition is satisfied. Also, the `vending_machine` object has to return all coins when `cancel` button is pressed. Although `customer` objects do not have explicit triggers, they have an interface with `vending_machine` objects and this allows them to require actions associated with the visible events. Thus `customer` objects are active objects as well.

4 Concurrent Logic Programming and $\mathcal{OASIS}$

Concurrent logic languages arise as an attempt to improve the efficiency of logic languages by exploiting stream *AND* parallelism. Besides, they are high level programming languages and convenient for parallel and distributed systems.

We are interested in modeling objects as perpetual processes according to the $\mathcal{OASIS}$ execution model. The identity of the object is the name of an input stream argument of the process. Works in this direction are principally based on making OO extensions to concurrent logic languages. Although implementation aspects are common, our motivation is different, we want to generate automatically a concurrent logic program corresponding to an $\mathcal{OASIS}$ conceptual model.

It is said that an object is implemented as a perpetual process because among the subgoals in which an object is reduced the same object appears. This produces the effect of continuity in the object life. Whenever the object goal is thrown as subgoal, in the reduction some of its attributes may be modified. Thus a change of state due to the occurrence of the associate action is represented. The effect "to execute an action" is obtained in the reduction when the new input channel is used as the original one without considering the last executed action. The formulae that define the change of state when the event is executed are subgoals that assign new values to the attributes of the object. In [9] a set of mappings between $\mathcal{OASIS}$ and KL1 has been established.

Following theses mappings, a translator from $\mathcal{OASIS}$ to KL1 has been implemented. This translator takes as input a system specification from an $\mathcal{OASIS}$ repository and produces a KL1 program that is compiled in order to obtain the prototype. The translator and the prototype are programs running in a Unix workstation. The interface has been implemented in Tcl/Tk using the Tcl plug-in for Netscape.

In $\mathcal{OASIS}$ the system behavior is determined by the behavior of its objects. An object behavior can be observed by analyzing the actions occurred and the states reached by the object. Thus, the animation of an $\mathcal{OASIS}$ specification allows examining actions and states of the objects.

Figure 3: An animation session.

Fig.3 shows the interface of a prototype of our animation environment. The object society is drawn in the upper left area. Objects are selected by clicking on them. On the right the traces of actions of an object (or object group) are listed. The list of traces can be filtered according to the kind of actions (sent, received, in conflict, executed or rejected). In the state area the state of the object is presented (only when one object is selected). Buttons *play*, *pause*, *stop*, *forward* and *review* are provided in order to control the session of animation. When the animation is paused it is possible to explore the traces of actions and states at previous instants. Eventually, the two entry widgets allow building an external action sent by the analyst in representation of one object in the system.

5 Petri Nets and $\mathcal{OASIS}$

"Petri Nets are a graphical and mathematical modeling tool applicable to many systems. They are a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic and/or stochastic" [11]. Due to the mathematical background of Petri Nets, all aspects related to formalization are more feasible. Also, analysis techniques can be applied to get properties of liveliness, etc. The enhancements of Object Oriented Petri Nets (OOPN) include for instance token values to be identifiers, inheritance, test and inhibitor arcs, etc. Tokens can encapsulate the activity determined by another Petri Net. OOPN have a single hierarchy that includes both token types and subnet types, thereby allowing multiple levels of activity in the net. Furthermore, OOPN provides functions to evaluate the state of the net without changing its state. The possible use of superplaces and supertransitions makes OOPN suitable to model both synchronous and asynchronous interactions between objects. Each instance of an OOPN class could be an independent object passing on messages between objects through tokens. The notion of OOPN superplace permits synchronous interaction. One transition deposits (or extracts) a token in (from) the superplace. This operation is synchronous with an internal transition of the superplace that accepts (or produces) the token.

We have experimented using the CodeSign tool [3] in order to implement and animate $\mathcal{OASIS}$ specifications by implementing its execution model. How the main features of the $\mathcal{OASIS}$ can be naturally and directly represented in OOPN has been analyzed [13]. Object Oriented concepts (such as classes, instances, interaction between objects, encapsulation, etc.) and spe-

cific $\mathcal{OASIS}$ concepts (such as preconditions, valuations, triggers, protocols, and operations) have been addressed by using the properties of the OOPN.

The support of multiple levels of activity in the OOPN makes the design of architectures easy enough to prototype a society of concurrent objects. Now we are working to obtain CodeSign code automatically from an $\mathcal{OASIS}$ specification.

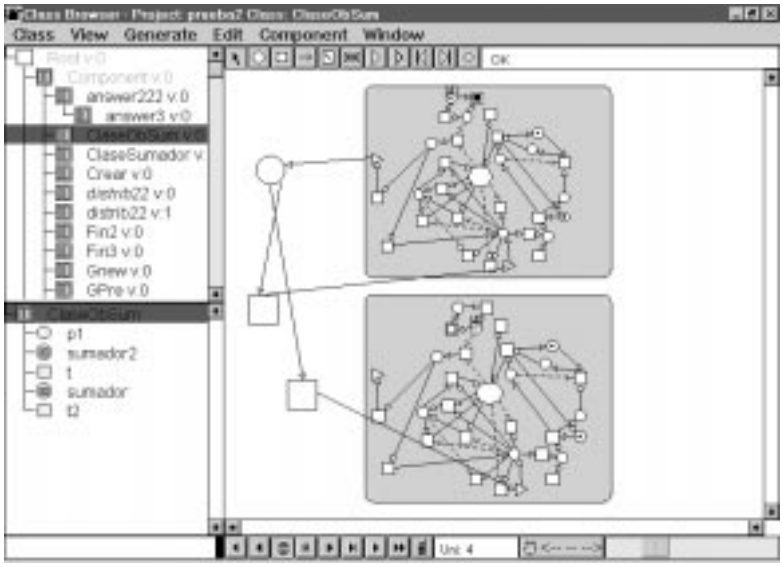

Figure 4: CodeSign Environment.

Fig.4 shows two $\mathcal{OASIS}$ classes (the grey boxes on the right) implemented in CodeSign using the Petri Net model. The bigger place in each class is the container of the instances of that class (notice that every object will have its own Petri Net instantiated.). The rest of the net is needed to direct the arrived actions towards the server object. The external net fragment is needed to establish the communication between the instances of both classes.

6 Conclusions

The main features of the $\mathcal{OASIS}$ language can be more naturally and directly represented in a concurrent environment. Using the execution model of $\mathcal{OASIS}$ as a guide we can obtain a useful animation of the $\mathcal{OASIS}$ specification. Our animation is only applied to purposes of requirements validation and do not claim to be the final software product.

We have built a translator program to obtain automatically both a concurrent logic program and a OOPN from $\mathcal{OASIS}$ specifications using the established correspondences. This work is being integrated into a CASE tool for system modeling supporting the $\mathcal{OASIS}$ approach. We have addressed the object-oriented concepts, and although we have focused on $\mathcal{OASIS}$, it could be extended to other similar languages.

The fidelity of the obtained concurrent implementation in relation to the $\mathcal{OASIS}$ system specification is a matter that is still being studied. In this case the verification and demonstration tasks would be supported by three important factors: firstly at the conceptual level the model is described in a formal language, secondly the abstract execution model is inspired by the semantics of that formal language, and eventually, concurrent programming languages and Petri Nets have a formal base. These factors do not determine the required justification for the translation process but give a way of formalization on which we are working.

We are currently considering the incorporation

of advanced concepts such as complex classes and complex communication mechanism.

References

[1] T. Chikayama. KLIC User's Manual. Institute for New Generation Computer Technology, Tokyo JAPAN, 1995.

[2] T. Conlon. Programming in PARLOG. Addisson-Wesley, 1989.

[3] R. Esser. An Object Oriented Petri Nets Approach to Embedded System Design, PhD Thesis, Swiss Federal Institute of Technology, Zurich, 1996.

[4] D. Harel. Dynamic Logic. In Handbook of Philosophical Logic II, editors D.M.Gabbay, F.Guenthner; pages 497-694. Reidel 1984.

[5] P. Heymans. The Albert II Specification Animator. Technical Report CREWS 97-13, Cooperative Requirements Engineering with Scenarios, http://sunsite.informatik.rwth-aachen.de/CREWS/reports97.htm.

[6] R. Herzig and M. Gogolla. An animator for the object specification language TROLL light. In Proc. Colloq. on Object-Orientation in Databases and Software Engineering, Montreal 1994.

[7] C. Lakos, From Coloured Petri Nets to Object Petri Nets, Proceedings of the 16th International Conference on the Application and Theory of Petri Nets, LNCS 935, Torino, Italy, Springer-Verlag, 1995.

[8] P. Letelier, I. Ramos, P. Sánchez and O. Pastor. OASIS 3.0: A Formal Approach to Object-Oriented Conceptual Modeling. SPUPV-98.4011, Servicio de Publicaciones Universidad Politécnica de Valencia, 1998. (in spanish).

[9] P. Letelier, P. Sánchez , I. Ramos. G. Prototyping a requirements specification through an automatically generated concurrent logic program. Gupta (Ed.) Practical Aspects of Declarative Languages, Lecture Notes in Computer Science LNCS 1551, pp. 31-45, Springer-Verlag, 1998.

[10] J.-J.Ch. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. In Notre Dame Journal of Formal Logic, vol.29, pages 109-136, 1988.

[11] T. Murata. Petri Nets: Properties, Analysis and Applications. Proceeding of the IEEE, Vol. 77, n^{o}4, April 1989.

[12] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois and P. Heymans. A Proposal for a Scenario Classification Framework, Technical Report CREWS 96-01, http://sunsite.informatik.rwth-aachen.de/CREWS/reports96.htm.

[13] P. Sánchez, P. Letelier and I. Ramos. Constructs for Prototyping Information Systems using Object Petri Nets, Proc. of IEEE International Conference on System Man and Cybernetics, pages 4260-4265, Orlando, USA, 1997.

[14] J. Siddiqi, I.C. Morrey, C.R. Roast and M.B. Ozcan. Towards quality requirements via animated formal specifications. Annals of Software Engineering, n.3, 1997.

[15] Wieringa R.J. and Meyer J.-J.Ch. Actors, Actions and Initiative in Normative System Specification, Annals of Mathematics and Artificial Intelligence, 7:289-346, 1993.