Using the COOPN/2 Formal Method for Groupware Systems Engineering#

Olivier Biberstein*, Didier Buchs+ and Nicolas Guelfi+

+ *Swiss Federal Institute of Technology*
Software Engineering Laboratory
CH-1015 Lausanne Ecublens
Switzerland

email: {Didier.Buchs, Nicolas.Guelfi} @ di.epfl.ch
phone: +41 (21) 693.67.68 - fax: +41 (21) 693.50.79

* *University of Geneva*
Centre Universitaire d'Informatique
24, rue du Général Dufour
CH-1211 Genève 4
Switzerland

email: Olivier.Biberstein@cui.unige.ch
phone: +41 (22) 705-7678 - fax: +41 (22) 705-7780

ABSTRACT. COOPN/2 *(Concurrent Object Oriented Petri Nets) is an object-oriented specification formalism based on Petri Nets and algebraic specifications. It is fully adapted to the specification of complex concurrent systems because it inherits the advantages of algebraic specifications for the specification of abstract data types, from Petri nets to describe partially ordered events and from the object-oriented paradigm for the 'structuring' of the specification. All these features are especially useful when speaking about groupware systems, which are application which demand rich modeling capabilities on concurrency and structuring. This paper explains the major principles for the specification of groupware systems with COOPN/2. This is done by mapping groupware concepts into COOPN/2 concepts and is demonstrated by means of an example of shared distributed diaries. Furthermore, we present the work done on the methodological approach for the development of concurrent object-oriented systems with COOPN/2 which cover analysis, specification, design, prototyping and testing.*

KEYWORDS. *groupware, replication, object-orientation, distributed systems, formal methods, Petri Nets, algebraic specification of abstract data types, software engineering.*

1. Introduction

This paper explains, by means of an example, how our concurrent object-oriented specification language (COOPN/2: Concurrent Object Oriented Petri Nets) can cope with some of the major groupware concepts. For this it is necessary to present important features of groupware systems as well as some main aspects of COOPN/2 in an abstract form. CO-OPN/2 is an object oriented specification language based upon Algebraic Petri Nets. Class and objects are key notions in the formalism of the object oriented part and relations between classes are given either by the inheritance mechanism for the reuse of properties description or by subtyping for the description of semantic preserving relations between classes. The use of Petri nets provide powerful concurrency modeling by means of the description of causal relations between events as well as a synchronization mechanism which allows for the linking of events together in order to build hierarchies of abstractions.

Since heterogeneous distributed systems aspects are fundamental in groupware, the paper focuses on these and interested readers will find more complete information on COOPN/2 in [5][13] and in the precursor article [7].

In the final part we present briefly the other features of the COOPN/2 environment such as the prototyping techniques and the testing techniques which should be used when software engineering techniques are to be considered within the groupware system development process.

2. Groupware

Cooperative work application development is by definition[1] a discipline which is related to:

- Distributed systems: the design of a groupware system involves many users on heterogeneous platforms which are linked through a computer network;
- Concurrency: the major interactive concept of the graphic interface is the representation of independent actions on specific resources;
- Networks and Telecommunications: both are directly dependent on the possibilities of the underlaying networks;
- Computer-Human Interaction: the design of a multi-user interface needs the development of new interaction modes.

Basic notions of groupware are shared context, view (of a shared context), user, synchronism or asynchronism (related to interacting time constraints), floor-control systems, data granularity, strong or loosely coupled works (dependent on amount of cooperation between users). The classical groupware systems include electronic mail, shared editors (Griffon [10], GroupDraw [12]) and desktop conferencing, among others.

Several general multi-layer models have been given in order to offer a rigorous framework for the development of groupwares [16], [14] or [11]. These approaches offer the possibility of adequately defining, for each layer, the sharing and instantiating

[1] following C.S. Ellis, a cooperative systems is a computer based system that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment.

This work has been partially supported by the Long Term Esprit Project DeVa (Design for Validation) number 20072 with the financial support of the OFES (Office Fédéral de l'Education et de la Science), contract number 95.0526 and by the Swiss National Science Foundation (FNRS) project on 'Formal Methods for Concurrency', grant number 2000-40583.94.

operations. For example, the SLICE model of [14] defines the following layers: abstract document, document representation, direct document manipulation, view representation, view manipulation, indirect manipulation and cursors. In our case, we consider only the first and the second layers, which cover the major notions.

Document manipulation is divided into two major modes: direct (resizing a figure by acting on one of its anchors), indirect (resizing a figure by defining some parameters in a dialog box) and shared. Each one can have a shared mode for some objects in a document and allow for the global representation of the users' manipulations.

The architecture of synchronous groupware systems must be studied in order to build a robust, transparent and rapid system and to allow for the modification of distributed data in satisfactory delays for the user without any constraint upon the graphic interface. The three major approaches for groupware architectures are the centralized, replicated and hybrid architectures. The first one involves only one process for managing all user accesses, the second introduces one process for each user and the processes communicate with each other in order to preserve the integrity of the data which are replicated on each site, and the third have a central process for managing the data integrity as well as a process for each user in charge of all non semantic actions (window managing for example) .

We will see in the following sections how CO-OPN/2 is well-adapted to the development of groupware systems by providing adequate structured modeling capabilities.

3. COOPN/2

COOPN is a specification language destined for the specification and the design of large concurrent systems. The most recent version, COOPN/2, extends the COOPN language by means of some object-oriented features such as class, inheritance and sub-typing.

3.1 Principles

The object-oriented paradigm is introduced into COOPN/2 in order to solve the problem of large specification management. A CO-OPN/2 specification corresponds to a collection of objects grouped into class templates used for instantiating the objects which compose the modeled system. A COOPN/2 object is a set of encapsulated state variables or attributes which can be used for specifying methods visible from the outside. Moreover, object reaction is modeled by means of the definition of internal transitions. According to common intuition, each object has its own behavior and evolves concurrently, along with the others. As the purpose of COOPN/2 is to capture the abstract concurrent behavior of each modeling entity, the concurrency granularity is not found in the objects but rather in the methods. Intra object concurrency can be obtained by expressing the causality relationship between the behavior of the different methods. The relations between objects, such as objects synchronization, may also be captured by using method synchronization schemes which allow for the construction of hierarchies of abstraction.

The concurrent behavior of the objects must be represented operationally. We propose the use of Petri nets, which are an elegant model of true concurrency (true in opposition to interleaving semantics). Unfortunately, standard Petri nets have very weak modeling capabilities in terms of their functional aspects and structuring capabilities. We add to Petri nets a synchronization scheme, modeling the possible concurrent processes needed by a method or an internal transition which might occur. This approach generalizes other approaches such as, the transition fusion or the Petri net systems.

Real systems must include other dimensions such as data structures, with associated operations acting upon these data. This is the reason why data structures are incorporated into Petri nets, forming a new class of Petri nets called high-level nets, i.e. Petri nets with tokens modeled by data structures instead of anonymous tokens. Abstract modeling is imposed for considering the logical formalism for the description of data structures. In our case, we have chosen algebraic abstract data types which combine abstraction with a simple notation, derived from the first order logic.

Thus, the idea of extending algebraic specification leads to a large number of approaches that may be classified into two main groups. A pure algebraic approach, which includes for instance the works of SMoCLS [1] as well as a mixed approach (property and model oriented) which includes the High Level Petri Net approach (algebraic nets [17], LOOPN [15], Cooperative Nets (CN) [19], OBJ-SA [3], CLOWN [4]).

3.2 The Architecture of a COOPN/2 Specification

The COOPN/2 formalism has adopted the object-oriented paradigm. A COOPN/2 specification is a collection of objects which interact concurrently. The notion of class allows one to specify a collection of objects which have the same architecture. Inheritance is a syntactic mechanism which avoids the development of classes from scratch and is mainly used for reuse specifications. A class may inherit all the features of another class and may also add some services or change the behaviors of some already defined services. However, a sub-typing relationship between classes implies a semantic relationship over class models. In other words, whenever a super-type object is used, a sub-type object may be substituted for it. Thus, all the behaviors of a super-type have to be preserved by the sub-type. Both inheritance and sub-typing hierarchies do not necessary coincide.

A COOPN/2 specification consists of two different kinds of modules: the abstract data type modules and the class modules. These two kinds of modules are composed of an interface, termed the module signature, and a body which primarily encapsulates the operations properties or the behavior of the objects. The abstract data type modules concern the data structure part of the specifications, and order-sorted algebraic specifications are used when describing these modules. Furthermore, the class modules correspond to a notion of encapsulated entities that possess an internal state and provide the exterior with various services. For this second sort of module, algebraic net formalism has been adopted. Objects can be statically or dynamically instantiated from the class modules.

For the class modules, two types of transitions are provided. On the one hand, internal transitions which correspond to object reaction are provided, and, on the other hand the parametrized external transitions, or methods, exist which represent the events visible from the outside. Cooperation between the objects is realized by means of a synchronization mechanism, i.e. each object event may ask to be synchronized with the methods of one or a group of partners by means of a synchronization expression. Four synchronizations operators have been provided. ***with*** for abstraction, ***'//'*** for simultaneity, ***'..'*** for sequence and ***'+'*** for alternative or non-determinism. Thus, a behavioral axiom is established in the form "*Event* ***with*** *Sync: Cond => Pre -> Post* in which *Event* is an internal transition name or a method name with parameters, *Sync* is the optional synchronization expression and *Pre*, *Post* describes the flow

relationship. For example the behavioral axiom "f(x) **with** (f1(x)**..**f2(x))//(f3(x)+f4(x)) **:** x>0 **=>** p(x) **->** p(x+1)" indicates that the parameterized event f(x) can occur if the behavior (f1(x)..f2(x))//(f3(x)+f4(x)) can be executed by the system with x a positive value of the place p. Its execution replaces x by x+1 in the place p and abstract the execution of the synchronization expression (f1(x)..f2(x))//(f3(x)+f4(x)) which means that f1(x) followed immediately by f2(x) is executed simultaneously with either f3(x) or f4(x).

Within the COOPN/2 framework, each class instance has an identity which is used as reference. Moreover, a type is explicitly associated with each class. Thus, each object identifier belongs to one type. In fact, the object identifiers are order-sorted algebraic values of a special order-sorted algebra, provided in order to capture the notion of sub-typing. This object identifier algebra is built so as to reflect the sub-type relationship between the classes, i.e. two carrier sets are related by inclusion if and only if the two corresponding type are related by sub-typing.

Algebraic sorts are defined together with related operations, called functions, in abstract data type modules. A variable of an algebraic sort has no identity. Algebraic sorts are used to specify values such as the primitive sorts (integer, boolean, enumeration types, etc.) or the purely functional sorts (stacks, etc.).

4. COOPN/2 for Groupware Engineering

In this section we explain first, by means of an example, how the COOPN features introduced in section 3 may be adapted to the development of groupware systems, as presented in section 2. Then we introduce briefly other aspects of the COOPN methodology such as prototyping, simulation and testing, which are useful for groupware systems engineering.

4.1 Case Study

4.1.1 Informal Specification

We define a simple shared diary for members of a 'software engineering laboratory' with a replicated architecture. This diary allows the following services: consultation, addition, cancellation and replacement of events. An event is simply composed of a day, a beginning and ending time and a comment line. The screen representations of a diary can be different between sites (graphical, textual ...).

Each site has four components for the diary of each member (see fig. 1): the Abstract Document Representation (ADR), the Distributed Synchronization Algorithm (DSA), the Data Access Layer (DAL) and the Graphical Interface Layer (GIL). All the laboratory members have their diary replicated on all the sites of the network (see fig. 2).

ADR keeps all information concerning the diary definition.

- DSA ensures that the replicated information is consistent.
- GIL is used for the graphical representation and handling of the diary by the host user.
- DAL filters accesses to the ADR.

The following sections explains in details the COOPN/2 classes which are defined for each of these component. All these classes will be instantiated on each host h as many times as there are opening requests concerning the user i of host h for interacting with the diary of the member j. Each instance will behave concurrently and each method could behave concurrently if the nature of the services does not imply a sequencing due to the unique events list. Each class instance could be seen as a server or as a client with respect to its role reported to the other instances.

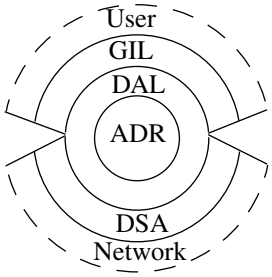

Fig. 1. Hierarchies of specification modules

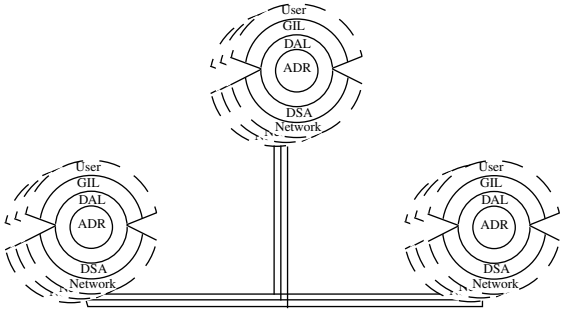

Fig. 2. Interaction of groupware subsystems

4.1.2 ADR: Abstract Document Representation

A diary is defined as a list of events. An event e is a tuple <d,bt,et,com> composed of the day, the begin time, the ending time and a comment. In COOPN/2 we define an algebraic specification for all these data structures (eventlist, event, day, time, comment). We define in COOPN/2 a class ADR which uses these data structures such that all events are in a list which is a resource considered in COOPN/2 as a token and put in a place diary of the class.

The ADR class must allow for the following functions:

- Consultation of the entire document: the method Consult is defined within the class ADR and has one parameter of type eventlist. This method reads only the list which is included in the place diary.
- Addition of a new event into the diary: the method AddEvent is defined in the class ADR with one parameter of the type Event. It adds the event given in the parameter on the list of the place diary.
- Cancellation of an event of the diary: the method Cancel is defined and has one parameter of type event which changes the list of the place diary in order to delete the event given in parameter.
- Replacement of an event by a new event: the method Update is defined with two parameters of the type event. In the event list of the place diary it replaces the first event parameter by the second.

4.1.3 COOPN/2 Code Example for the Class ADR

We give, in the graphical representation of figure 3 the graphic representation class ADR outline, which is defined over a list structure used for storing the events in the object.

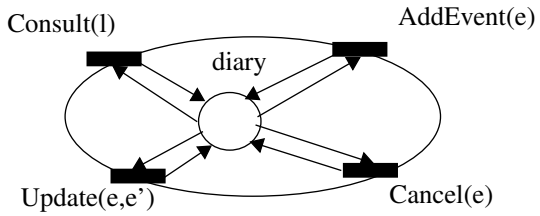

Fig. 3. An outline of ADR class

This structure impose a sequential access to the ADR instances. The corresponding textual representation of that class is given in figure 4. A comment for the method Update (*) could be: for every event e and e' and list of events l, the method Update can be executed with e and e' as parameters if (e is in the list l) and (l is in the place diary). Its execution verifies that l is removed from the place diary and the list (e'+(l-e)) corresponding to the original list with e replaced by e'. An incomplete algebraic specification for the listevent sort is given in figure 5.

```
Class ADR;
Interface
    Use ListEvent, Event;
    Methods
        Consult _ :        listevent;
        AddEvent _ :       event;
        Update _ _:        event, event;
        Cancel _ :         event;
Body
    Places
        diary: listevent;
    Axioms
    Consult(l) :      diary l -> diary l;
    AddEvent(e) :     diary l -> diary( l + e);
(*) Update(e, e'):    e isin l = true =>diary l ->diary (e'+( l-e));
    Cancel e :        e isin l = true => diary l -> diary ( l - e);
Where   e, e' : event;
        l : listevent;
End ADR;
```

Fig. 4. A textual representation of the Class ADR

```
Generic Adt ListEvent;
Interface
  Use  Naturals, Booleans, Event;
  Sort listevent;
  Generators
     []    : -> listevent;
     _ + _ : event listevent -> listevent;
  Operations
    _ + _ :         listevent listevent -> listevent;
    _ - _ :         listevent event -> listevent;
  isin _ _ :        event listevent -> boolean;
    _ = _ :         listevent listevent -> boolean;
Body
  Axioms
     ([] + l1)  = l1;
     ((e + l1) + l2) = (e + (l1 + l2));
     ...
Where      l1, l2 : listevent;
           e     : event;
End ListEvent;
```

Fig. 5. A textual representation of the sort listevent

4.1.4 DAL: Data Access Layer

The COOPN/2 class defined for this part must offer the following services:

- consultation of an event e in the diary: a method Consult is defined with one parameter of type event. This method, use a synchronization (method call) to look inside the diary which is guarded by the associated ADR class instance, if the event e is in the event list.
- the same three services provided by the class ADR which are Consult, AddEvent and Update and which are used both for interacting on the abstract document representation and for updating the graphical representation.

4.1.5 GIL: The Graphical Interface Layer

A class GIL is defined such that it can handle the document representation and manipulation taking into account the host graphical capabilities (both hardware and software).

This class has a method with a parameter of type event, in order to receive the display request for the parameter from the data access layer. It must have certain informations such as how to make a graphic representation of event on the screen. For example, an event could represented by a black box or by the comment text part. The various events could all be placed on a grid or only listed ... The method Interact is used for modeling the user's graphical interface manipulation of the document which induces a new event to be taken into account by the diary.

4.1.6 DSA: Distributed Synchronization Algorithm

This layer is crucial with respect to groupware system. It is in charge of the synchronization of the replicated data which concerns the diaries. Several algorithms may be studied and are entirely dependent upon the application type. Sophisticated principles may be applied in order to ensure synchronization based upon data granularity and the nature of the manipulations.

In COOPN/2 we define a class DSA with several data in state variables (events, logic time information ...) along with the two main services:

- Dispatch of an event e: a method dispatch is defined and is used by GIP in order to broadcast, if necessary, information modification to all hosts within the network. Notice that not all of the diary modifications are taken into account systematically. The algorithm may, for example, have in its queue some reversible actions.
- Receive an event e: a method Receive is defined and used by the network layer. Its implementation must ensure an analysis of the event in order to keep a coherent abstract document representation. Such notion like permutable events, reversible events, logic time control are managed at this level. Furthermore this method must generate a synchronization request with the data access layer for the updating of abstract document representation as well as graphical representation.

4.1.7 Individual Diary Environment

Finally we have introduced a class IndDiaryEnv in COOPN/2 in order to create an association between the instances of the four classes described above which are all necessary for management of the diary.

This class only defines an internal event which may be automat-

ically executed thus creating the necessary instances with an initialization phase which transmits the identities of the objects between the instances.

More precisely, it sequentially creates the instances gil, dal, adr and das. Then it initializes them: gil with the reference on dal, adr with dal, dsa with dal and dal with adr, gil and dsa.

In figure 6, we have given a partial representation of the objects, their synchronizations and their clientship relations. Inside each oval, which represents an encapsulated class, one Petri net describes the behavior of the class under consideration. This Petri net represents the state of the objects through the places (circle) and the object methods by means of parametrized transitions (black rectangle). Two types of transitions are available in COOPN/2. The first of these, internal transitions (the usual transitions of Petri nets) appear on the graph as white rectangles, and the second, external transitions, or methods, are represented as black rectangles place on the border of the class circle. These transitions can be parameterized so as to carry information from the sender to the receiver. Synchronizations between transitions are drawn using grey arrows.

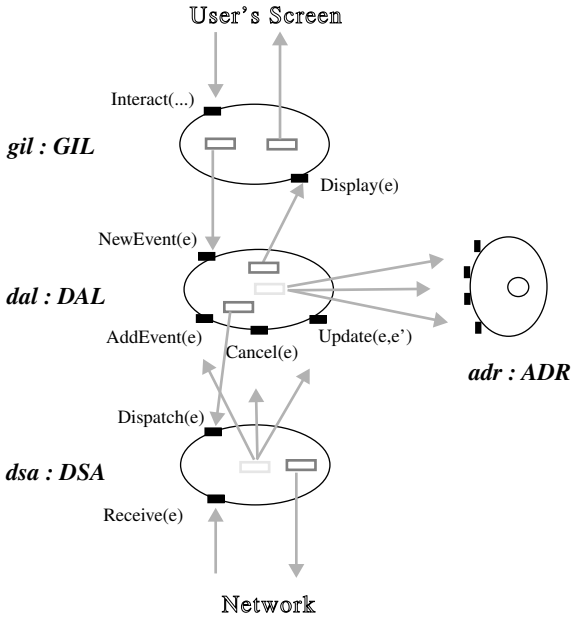

Fig. 6. Class Instances for an Individual Diary System

4.2 Managing the Software Life-cycle

In the following subsections we present in a general way our considerations for managing the software process with COOPN/2. Some examples are given considering groupware development.

4.2.1 Analysis

The COOPN/2 formal specification language cannot address the very first step of the software process. We believe that this step, called analysis, should be done using an intuitive and 'natural' language adapted to the description of computerized systems. The analysis is well adapted to be followed by a formal specification step in order to obtain a precise (unambiguous and consistent) description of what the future system should do.

Considering analysis methods which are based on object oriented principles, we can cite the Object Modeling Technique (OMT) [18] or Fusion [9]. In fact, these methods can cover the analysis phase as well as the design and implementation. Our idea is to use the language and some principles defined in these methods for the analysis phase and to systematically derive a formal specification using COOPN/2. Such work has been done for the link between a subset of OMT and VDM++ [20]. We have started a project in order to connect the Fusion analysis phase and COOPN/2.

4.2.2 Refinements Principles

One of the characteristic of COOPN/2 is its ability to easily allow for the evolution or refinement of distributed concurrent entities. Several examples of these aspects are given below.

If we wish to obtain various graphical interfaces between platforms, we should change the GIL class definition in order to allow for a heterogeneous distributed system. For example, the GIL class is represented differently for the UNIX, WINDOWS or MacOS environments, respectively. Nonetheless, in COOPN/2 all these classes could actually be sub-types of the initial GIL class as they have the potential to display events.

The same reasoning is valid for the synchronization algorithm. New classes could be easily substituted for the class DSA in order to evaluate other algorithms or to take into account new communication protocols within the network layer.

Concerning a laboratory diary, we could use DSA, ADR and DAL without any changes and introduce a new class GIL'. This class GIL' would be derived by inheritance from GIL and would add information and services devoted to displaying information on the access periods of the laboratory (for example, in red on the graphical representation). This information would only be used as free information without having any consequences on the ability to put an event into an access denied period. Thus, subtype relation between GIL' and GIL is allowed to exist. The same principle could be applied to any other part of the system and the subtyping relation would be guaranteed based upon behavior preservation.

4.2.3 Simulation and Prototyping

The development of applications in COOPN is supported by the SANDS (Structured Algebraic Net Development System) [6] environment which includes both simulation tools and prototyping tools. The simulation tool is based upon the same techniques as those of the prototyping tool with the exception that the computations are symbolic and are performed on an higher level of abstraction.

Our proposal for prototyping [21] consists of a combination of incremental and throw-away prototyping as it may evolve by virtue of a process refinement beginning in the early stages in the software life cycle and continuing until its fully implemented version. The code generated by the system's compiler can form a skeleton representing the application, or of a particular module in the application, and should therefore be efficient and legible enough. Generated modules can be incrementally substituted by more efficient hand-written code.

4.2.4 Testing

Writing programs from specifications is of course necessary in order to create a realistic software systems. Nevertheless, because of the potential for producing unsound or incomplete realizations it is interesting to develop a way to identify possible fallacies which might be created during writing. Testing methods are proposed essentially in order to detect potential errors without having to prove concretely the completeness of the pro-

grams with respect to the specifications.

We have introduced in [8] a methodology for testing concurrent programs against their formal specifications and, in particular COOPN/2 specifications (Petri Nets and Algebraic specifications). In this specific case, we explore the testing of units against their formal specifications by means of sound principles for selecting adequate for selecting adequate test sets. This seems to be one of the most promising approaches for allowing the implementation of critical systems to be implemented in confidence. Our framework presents the principles that must be used in the selection of valid tests which are to be applied automatically to the concurrent programs and also helps to determine how well the tests are to succeed. It is obvious that these tests must be finite and that, as a consequence it is not possible to prove completely the adequation of the programs with respect to their specifications. This may be done only in a case in which certain hypotheses are to be verified. These hypotheses must represent current testing practices. Previous work on testing methods has been described using abstract data types [2] and has now been extended to the concurrent programs, in order to be able to test Ada programs which are specified using COOPN/2.

Concerning groupware systems, we are interested in test sets for ensuring the consistency of the behaviors produced by the distributed Synchronization algorithm for replicated data integrity.

5. Conclusion

We have shown that CO-OPN/2 can manage the development of groupware systems by adopting a layered structure which distinguishes the different groupware entities such as the user part, the protocol and the network. Refinement, testing and prototyping procedures are useful in development quality systems by means of a rigorous process that takes are into account the distribution configuration and inherent concurrency of cooperative applications.

6. Acknowledgments

We would like to thank Alain Karsenty for his helpful suggestions concerning groupware.

7. Bibliography

[1] E. Astesiano, G. Reggio, Algebraic Specification of Concurrency, in LNCS N° 655, pp 1 - 39, 1992.

[2] S. Barbey, M. Ammann, and A. Strohmeier. Open issues in testing object-oriented software. In Karol Frühauf, editor, ECSQ '94 (European Conference on Software Quality), pages 257– 267, Switzerland, October 17-20 1994. Also available as Technical Report EPFL-DI-LGL No 94/45.

[3] E. Battiston, F. De Cindio, G.Mauri, "OBJSA a class of High level nets having objects as domains", LNCS n°340, pp 20-43, 1988.

[4] E. Battiston, F. De Cindio, "Class orientation and inheritance in modular algebraic nets", In Proceedings International Conference on Systems, Man and Cybernetics, volume 2, pages 717--723, Westminster, Le Touquet-France, 1993.

[5] O. Biberstein and D. Buchs, 'Structured Algebraic Nets with Object-Orientation', Workshop on Object-Oriented Programming and Models of Concurrency'95, 1995, Turin.

[6] O. Biberstein, D. Buchs, M. Buffo, C. Buffard, J. Flumet, J. Hulaas, G. di Marzo and P. Racloz, 'SANDS1.5/ COOPN1.5 An Overview of the Language and its Supporting Tools', Technical Report 95/133, EPFL-DI-LGL, June 1995.

[7] D. Buchs and N. Guelfi. A concurrent object-oriented Petri nets approach for system specification. In M. Silva, editor, 12th International Conference on Application and Theory of Petri Nets, pages 432–454, Aahrus, Denmark, June 1991.

[8] D. Buchs. Test selection method to validate concurrent programs against their specifications. In SQM '95 (Software Quality Management), pages 403–414, Spain, April 1995. (Also Available as Technical Report EPFL-DI-LGL No 95/101).

[9] D. Coleman and P. Arnold and S. Bodoff and C. Dollin and H. Gilchrist and F. Hayes and P. Jeremaes", Object-Oriented Development: The Fusion Method, Prentice-Hall, 1994.

[10] D. Decouchant, V. Quint, I. Vatton, "L'Edition coopérative de document avec Griffon", actes des quatrièmes journées sur l'ingéniérie des interfaces homme-machine (IHM'92), 1992, pp137-142.

[11] C.S. Ellis, J. Waigner. A Conceptual model of Groupware, ACM CSCW'94, Conference on Computer Supported Cooperative Work, USA, 1994.

[12] S. Greenberg and M. Roseman and D. Webster and R. Bohnet,"Human and Technical Factors of Distributed Group Drawing Tools", Interacting with Computers, Special Issue on CSCW: Part 1, volume 4, number 3, pages 364--392, 1992.

[13] N. Guelfi, 'Hierarchical Algebraic Nets', Technical Report EPFL-DI-LGL n°95/99, January 1995. also in "Les Réseaux Algébriques Hiérarchiques" : un formalisme de spécifications structurées pour le développement de systèmes concurrents, PhD Thesis, Paris XI Orsay University, France, 1994.

[14] A. Karsenty, 'GroupDesign : un collectitiel synchrone pour l'édition partagée de documents', PhD Thesis, Paris XI Orsay University, France, 1994. Also in "GroupDesign: Shared Editing in a Heterogeneous Environment, Article in Computing Systems, vol. 6, no. 2, p. 167--192, 1993.

[15] C.A. Lakos, C.D. Keen, "LOOPN++: A New Language for Object-Oriented Petri Nets ", European Simulation Multiconference, Barcelona, 1994.

[16] M.A. Linton, J.M. Vlissides, P.R. Calder, "Composing User Interfaces with Interviews", IEEE Computer, pp 8-22, 1989.

[17] W. Reisig : 'Petri nets and Algebraic specifications', Theoretical Computer Sciences, n° 80, 1991.

[18] J. Rumbaugh and M. Blaha and W. Premerlani and F. Eddy and W. Lorensen, Object-Oriented Modeling and Design, Prentice Hall, 1991.

[19] C. Sibertin-Blanc, Cooperative nets, 15th International Conference on Application and Theory of Petri Nets 1994, Zaragoza, Spain, pages 471--490, LNCS n°815, 1994.

[20] VERILOG S.A., VENUS OMT-VDM++ Coupling Module User Manual , ESPRIT project 6500 'AFRODITE', document AFRO/VERILOG/PLB/UM/V3.0, 1995.

[21] D. Buchs and J. Hulaas, ``Incremental Object Oriented Implementation of Concurrent Systems based on Prototyping of Formal Specifications," SIPAR workshop, 1995, pp. 141-145, Switzerland.