A Coloured Petri Net Formalisation for a UML-Based Notation Applied to Cooperative System Modelling

José Luis Garrido and Miguel Gea

Dpt. Lenguajes y Sistemas Informáticos, University of Granada,
E.T.S.I. Informática, C/Daniel Saucedo Aranda s/n, 18071 Granada, Spain
http://giig.ugr.es
{jgarrido,mgea}@ugr.es

Abstract. New approaches are currently being adopted to address the development of cooperative systems, although not many standards exist that can be used to develop this type of interactive system. We apply the standard Unified Modelling Language (UML) notation within a methodology aimed at the analysis and design of such systems, and present a semantic formalisation of the UML notation used to model cooperative systems. The semantics and its application are described on the basis of translation schemes to Coloured Petri Nets and the benefits of formalisation are shown.

1 Introduction

To date, Computer-Supported Cooperative Work (CSCW) [15] has comprehended various systems: Workflow Management Systems (MfMS), computer-mediated communication (CMC) (e.g. e-mail), decision support systems, shared artefacts and applications (e.g. shared whiteboards, collaborative writing systems), meeting systems, etc. These can be categorised in several ways. One of these is by the function that the system performs, and another interesting one is based on two dimensions: time (synchronous or asynchronous) and space (co-located or remote). The outcome matrix is very useful to refer to the particular circumstances that a groupware application aims to address. Groupware [6,3] has been defined as *a computer-based system that supports groups of people engaged in a common task (or goal) and that provides an interface to a shared environment*. We note that in the set of fields and systems embraced by CSCW there are many explicit and implicit related concepts. These tend to either logical or technological aspects, such as interaction, communication, coordination, information, group, behaviour, distribution, control, etc. An additional difficulty is that the same term is frequently used in two or more fields but with different meanings and implications. For instance, human-computer interaction (HCI) particularly addresses psychological and computer issues; on the other hand, in human-human interaction within group activities, social issues acquire more relevance. An introductory study of the concepts involved in specifying the general principles and properties of CSCW systems is presented in [7].

The development of groupware applications is more difficult than that of a single-user application, because social protocols and group activities must be taken into account for a successful design [12], and so techniques aimed at enhancing group interaction activities should be applied. The inherent complexity of any interactive system requires a great deal of effort in the formulation of specifications, and formal methods may be used to achieve this goal [2,13]. At the modelling stage, support should be provided to study the system being developed.

The work presented in this paper is part of a new methodology called AMENITIES [9] (acronym for A MEthodology for aNalysis and desIgn of cooperaTIve systEmS). This methodology is based on behaviour and task models for the analysis and design of generic cooperative systems. The set of behaviour and task models (called Cooperative Model of AMENITIES) is a conceptual model which comprises the foundation of the methodology. The paper shows how to formalise the semantics for the Cooperative Model, which uses a UML-based notation, by describing translation schemes to Coloured Petri Net (CPN) formalism [14]. The paper is organised as follows. Section 2 reviews related work. Sect. 3 justifies the use of CPN formalism to define UML semantics. The conceptual framework for the Cooperative Model is introduced in Sect. 4 and an application example of this model is presented in Sect. 5. Then, on the basis of the example provided, the question is addressed of how, formally, to represent the model to be studied (Sect. 6). Finally, Sect. 7 summarises the main conclusions.

2 Related Work

Several approaches have been proposed to specify cooperative systems, focused on representing user tasks [19]. Thus, the system specification is a collection of user goals each of which is defined by the sequence of tasks that allow us to achieve a desired objective. Several notations have been proposed, such as GTA (Groupware Task Analysis) [29] and CTT (ConcurTaskTrees) [22]. GTA proposes an ontology-based system study for task world models, that is, a framework in which participants (agents and users' roles), artefacts (objects) and situations (goals, events) take place. Moreover, a set of relationships between these are clearly identified (uses, performed-by, play, etc). CTT provides a hierarchical graphical notation to describe concurrent tasks, and allows us to specify cooperation by adding a hierarchical specification with temporal constraints for each cooperative task. This extension and others aim to establish common tasks for several users and the relationships between them. An approach describing both task and system models by means of a dialect of Petri Nets (ICO formalism) is presented in [21], demonstrating how formal task models improve the design of interactive systems.

These task-based approaches study the system from the user's point of view, describing the cognitive skills required for correct use. However, most such techniques do not consider certain dynamic aspects in the problem domain [18]. For example, the user's role may change during real situations (e.g. the responsibilities in an office department), and the ways in which the objectives to be achieved may vary (e.g. a new commercial strategy, different work organisation). Thus, group organisation and evolution over time should be taken into account when social organisations are described.

On the other hand, our work is based on UML notation to describe cooperative systems by extending task-based approaches with ethnographic and cognitive issues. The advantage of this election is the use of a standard successfully adopted in software engineering. However, it is a semi-formal notation and various proposals have been made to formalise UML for different purposes, including:

– Stochastic Petri Nets, derived from UML statecharts and a collaboration diagram, for performance analysis [16].
– Validation of architectural software design with UML collaboration diagrams using CPN behavioural templates [23].
– Development of the CPN of a system by deriving Object Oriented Petri Nets Models from UML statecharts and connecting them using UML collaboration diagrams [26].
– Defining formal operational semantics by transforming UML state diagrams into graphs [11].
– Giving execution semantics for UML activity diagrams intended for workflow modelling [5].
– Definition of a semantics for collaboration and activity diagrams based on Place-Transition Petri Nets with informal inscriptions [10].

In general, these proposals focus on software analysis or toolkit construction for code generation, although some consider workflow modelling using activity diagrams.

3 UML Semantics versus CPN Semantics

The OMG Unified Modelling Language Specification [20] (in its current version 1.4) specifies syntax and informal semantics of the notations embraced by UML. This specification is clearly focused on software models. In particular, with respect to the behavioural UML notations that we use for the cooperative model of AMENITIES, the following text appearing in this reference should be highlighted:

1. *Statecharts.* "A statechart diagram can be used to describe the behaviour of instances of a model element such as an object or an interaction... The semantics and notation described are substantially those of David Harel's statecharts with modifications to make them object-oriented".
2. *Activity diagrams.* "An activity graph is a variation of a state machine in which the states represent the performance of actions or subactivities... It represents a state machine of a procedure itself".

A statechart (or activity diagram) is a graph that represents a state machine. The relationships between a state machine and its context have no special notation. The state machine also provides the semantic foundation for activity graphs. The specification document provides, for statecharts and activity diagrams, both a specification of the notation (graphic syntax) and an informal definition (in English) of their semantics.

The semantics of a UML state machine (i.e. the metamodel) is described in terms of the operations of a hypothetical machine that implements a state machine specification. Thus, this operational semantics is defined by the following key components:

- A queue of incoming event instances to be dispatched.
- The dispatcher mechanism for event processing.
- An event processor that processes dispatched event instances according to the general semantics and the specific form of the state machine in question.

Our main motivation in describing the semantics of UML notation applied to CSCW systems by using CPNs is that the existence of several variation points allows different semantic interpretations that might be required in different application domains. This is usually our case, and so high-level Petri Nets are used for the formal specification. This provides the following advantages:

- CPNs provide true concurrency semantics by means of the step concept, i.e. when at least two non-conflictive transitions may occur at the same time. It is the ideal situation for our application domain, as the model must be supported by a multithreaded state machine (several actors moving within the same space of states).
- The combination of states, activities, decisions, data, events and complex transitions (namely fork-join constructions) means that the UML state machine notation is very rich. CPNs allow us to express, in the same formalism, both the kind of system we are dealing with and its execution.
- Formal semantics is better in order to carry out a complete and highly automated analysis (i.e. validation and verification of properties) for the system being designed.

4 Framework

The cooperative model of AMENITIES, which is the core of the methodology, adopts an overall view of a system. Hence, the system embraces computer-based systems as well as the end-users themselves and related aspects such as organisation, communication, collaboration [27] and coordination [17]. This paper does not deal with the user's behaviour except that related to the interaction between different users. It is even possible to model interactions in which no computer-based system is involved [8]. The model allows us to carry out task analysis and modelling, as well as represent other related aspects such as roles, capabilities, constraints, etc. The notation used is basicly that of UML [25] only that instead of applying it to specify concrete classes for implementation, we consider other more abstract classes (group, role, ...) in the domain of cooperative systems. Because the emphasis is to study behaviour rather than structural aspects, class diagrams are not shown in this paper.

Several definitions of framework have been made, depending on the level where it is applied. In this context, a framework should give a higher common abstraction level between a family of related systems to be described in terms of general concepts. Thus, a framework is a pattern encompassing the principal common concepts of a kind of system and the relationships between them.

We define the basic terminology as follows. An *action* is a basic unit of work, executable atomically. A *subactivity* is a set of related subactivities and/or actions. A *task* is a set of subactivities intended to achieve certain goals. A *role* is a designator for a set of capabilities to carry out work, including tasks, skills, constraints and responsibilities/authorities. An *actor* is a user, program, or entity that can play a role in the execution of, or responsibility for, tasks. A *cooperative task* is one that must be carried out by more than one actor, playing either the same role or different ones. A *group* is a set of actors playing roles and organised around one or more cooperative tasks. A group may be composed, i.e. formed from related subgroups. A *constraint* (also called a *law*) is a limitation imposed by the system that allows it to adjust the set of possible behaviours dynamically. Finally, a *capability* is a constraint that directly affects an actor or group, enabling them to respond to new challenges offered by the system, as a result either of external events or of internal events produced by the interaction between participants (i.e. actors, groups or the system itself).

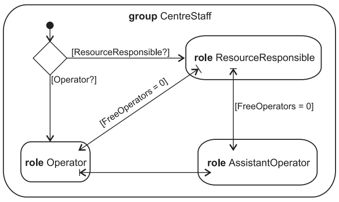
The method for building the cooperative model (as is shown in the next section) is based on two key concepts defined above: work and group. We use the notion of task to structure and describe the work that must be performed by the group. This provides the way to translate work, i.e. something that is tacit and implicit, into something that is concrete and explicit. Nonetheless, tasks are also considered at a very abstract level as noted above. A group, on the other hand, can be more or less explicit. Sometimes organisational aspects determine the way people work, but in other cases personal and/or operational aspects are the basis for organising people in order to perform an activity. The notion of role, in any case, allows us both to specify groups as needed and to establish dynamic relations between actors and tasks.

5 Example of Cooperative Model

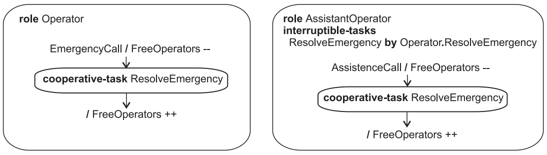
As an example to introduce the syntax and semantics of statecharts and activity diagrams of UML according to the application domain (i.e. cooperative work), we have modelled the current system used in Emergency Coordination Centres in Sweden and the U.S.A. [1]. These systems were designed and implemented fulfilling the control requirements of extreme situations. The Centre distributes tasks and is responsible, in the case of large-scale accidents, for the coordination of the organisations involved (police, fire brigade and medical help), until all units have arrived at the scene of the accident. At that point, the fire brigade takes over responsibility for coordination. The main goal is to assign and manage resources as fast as possible, as well as to assess the particular conditions of each emergency. The sequence of steps to build the cooperative model is shown in the following subsections.

5.1 Organisation Group

The first step in the method is to specify groups by means of UML statecharts, as shown in Figure 1(a) . This is based on identifying the related roles, there is one state for each role. The concept of role allows us to state the dynamic connections between actors and tasks. Thus, the basic structure of the organisation is described: actors play the roles `Operator`, `AssistantOperator` and `ResourceResponsible`. In this case, capabilities (e.g. guard `[operator?]`) initially determine which role is played by each actor, depending on his/her professional category, specialisation and/or skills. It also specifies (e.g. guard `[FreeOperators=0]`) under which constraints dynamic behaviour changes must be produced, sometimes as a result of interactions between members of the group.



(a)


role Operator
EmergencyCall / FreeOperators --
cooperative-task ResolveEmergency
/ FreeOperators ++
role AssistantOperator
interruptible-tasks
ResolveEmergency by Operator.ResolveEmergency
AssistenceCall / FreeOperators --
cooperative-task ResolveEmergency
/ FreeOperators ++

(b)

Fig. 1. (a) Group definition (b) Role definitions

5.2 Role Definition

Actors' knowledge of the system is determined by the functionality of the latter. In Fig. 1(b) the tasks that can or must be performed are identified. Thus, each role is actually a UML composed state including a submachine for each task that can/must be performed. Here, the defined roles collaborate on the single task `ResolveEmergency`. For the role `AssistantOperator`, the task being performed can be interrupted (section `interruptible-tasks`) if there is a new emergency.

In this situation, the actor will behave as `Operator` in order to respond to this new emergency.

5.3 Task Definition

The next step describes by means of UML activity diagrams the subactivities/actions needed to carry out each task (Fig. 2). By means of sequential (arrows) and concurrent (thick bars) constructions, temporal- ordered constraints of subactivities are specified. Diamonds may also be used to specify decision points involved in certain strategies during task performance. For each subactivity or action, the task definition includes specifying those responsible and the optional roles needed to accomplish it. Optional roles are shown between brackets and the symbol '|' specifies an inclusive-or relationship. This role specification is an extension to UML swimlanes for activity diagrams.

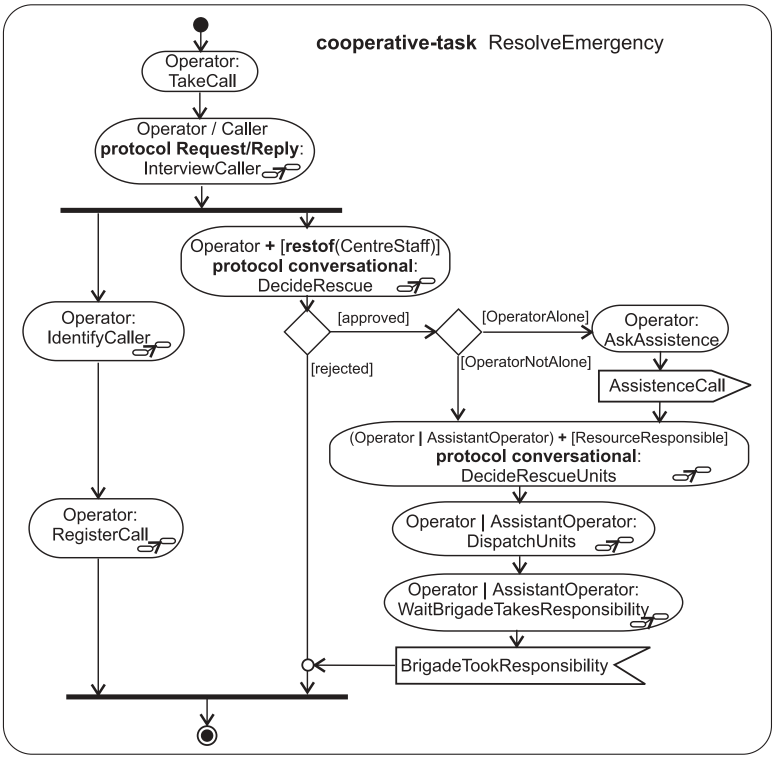

cooperative-task ResolveEmergency
Operator: TakeCall
Operator / Caller protocol Request/Reply: InterviewCaller
Operator + [restof(CentreStaff)] protocol conversational: DecideRescue
Operator: IdentifyCaller
[approved]
[rejected]
[OperatorAlone]
[OperatorNotAlone]
Operator: AskAssistence
AssistenceCall
(Operator | AssistantOperator) + [ResourceResponsible] protocol conversational: DecideRescueUnits
Operator: RegisterCall
Operator | AssistantOperator: DispatchUnits
Operator | AssistantOperator: WaitBrigadeTakesResponsibility
BrigadeTookResponsibility

Fig. 2. Task Definition

5.4 Specification of Interactions between Actors

The above task definition includes several tasks to be carried out by means of interaction protocols. For instance, the subactivity `InterviewCaller` specifies the type of protocol `Request-Reply` that should be used to accomplish this, as well as the participants involved: `Operator` asks and `Caller` answers. On the order hand, the activity `DecideRescueUnits` specifies a conversational protocol, i.e. any participant can take part in this activity in any order and with the same degree of responsibility.

6 Semantics for the Cooperative Model

This section states the foundation of a specific semantics for the cooperative model. The idea is to be able to automate the behavioural analysis for the cooperative system by making explicit concurrency, non-determinism, synchronization and resource sharing. CPN can be sujected to various Petri Nets analysis techniques which aid in the validation of UML behavioural specification.

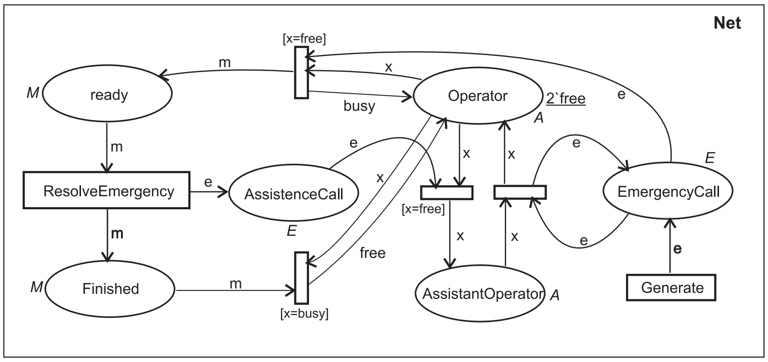
6.1 Translation Schemes from UML to CPN

In the following subsections, we discuss the key points in formally defining notation semantics according to the application domain. This is obtained by translating elements of the cooperative model to CPN components, as shown in Fig. 3, which is the corresponding CPN for the above modelling example. The subnet `ResolveEmergency` in Fig. 3(b) is for the transition with the same name in Fig. 3(a).

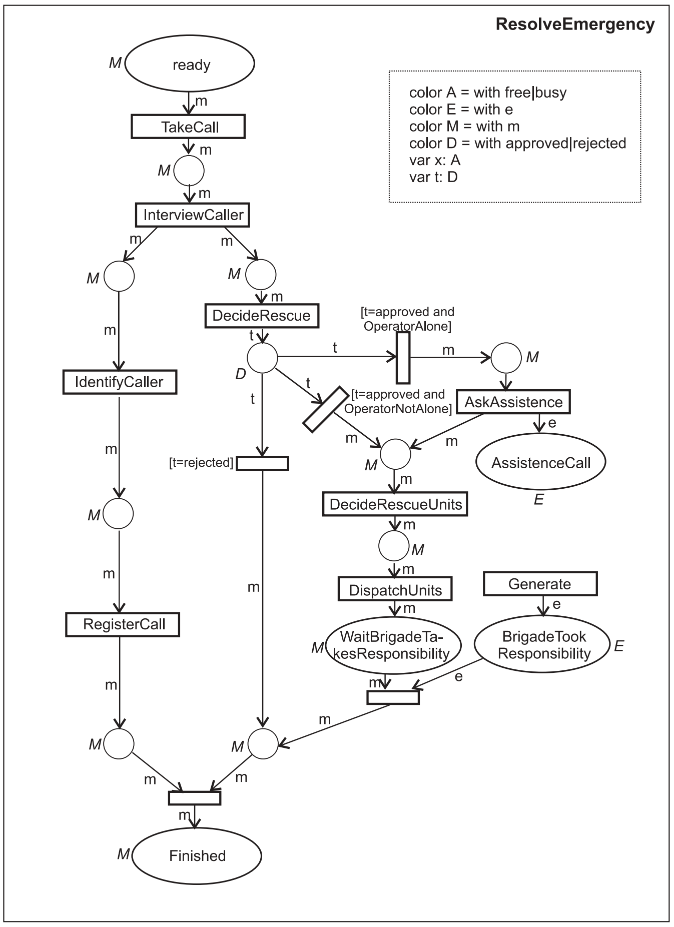
State/Activity Mapping. There exist two general alternatives to apply transformations from the cooperative model to the CPN model:

1. Identify subactivities with places in the CPN, allowing them to be interrupted (if necessary) in a direct way. In a cooperative system, there are both atomic and non-atomic activities, subactivities and actions respectively. Subactivities can be interrupted (if specified), such that the actors playing them may leave and return to the subactivity later. For example, the actor playing the role `AssistantOperator` in Fig. 2 can abandon the subactivity `DecideRescue` during its performance.
2. Conversely, the other possibility is to identify subactivities with transitions. At first glance, no activity could be interrupted since transition firing is considered instantaneous in Petri Nets.

We have chosen the second alternative, for several reasons. First, mapping subactivities into places poses the following problem: if a place represents a subactivity state, when the actor returns the subactivity will start again. Thus, to represent the leaving point where a subactivity continues would be impossible.



(a)



color A = with free|busy
color E = with e
color M = with m
color D = with approved|rejected
var x: A
var t: D

(b)

Fig. 3. CPN for the Emergency Coordination Centre

Secondly, the common Petri Net hierarchical modelling technique is by means of substitution transitions, and therefore if a transition represents a subactivity, there always remains the possibility of decomposing it into various actions (other transitions) and resting points (places) that enable interruptions and returns. This is also the ideal solution to the previous problem. Thirdly, modelling subactivities by transitions allows us to model data flow in the places of the subactivity flow more clearly (e.g. place of type `D` in Fig. 3(b)).

Actors and Resources. As shown in the CPN model in Fig 3(a), a role is considered equivalent to a type of resource, which is represented in a Petri Net as a place and a specific type associated (type `A`). Hence, there would be one token in the place for each actor playing this role. Each one of these places is labelled with the corresponding role name. For instance, there is a place labelled `Operator` with a token type `A` and, initially, two `free` value tokens (inscription `2‘free`). For the sake of simplicity, the net does not include the role `ResourceResponsible`, and hence does not include the transition to take the initial decision corresponding to the diamond in Fig 1(a). The arcs from role places to the subactivity transitions and vice versa, corresponding to the roles performing activities in Fig. 2(b), have also been omitted.

Guards. In CPN, transitions can have guards associated, which implies that the UML guards have a direct translation to CPN models, i.e. while the guard condition is not satisfied the transition cannot occur. Guards expressed in natural language must be interpreted. This is done in part (Fig. 3(b)) for the two consecutive decision points in Fig. 2. These are merged in the CPN, producing as many transitions as the total number of outgoing paths. The first condition (`approved` or `rejected`) in the guard is translated on the basis of a specific piece of information outgoing from the activity `DecideRescue` (type `D` token). With respect to the second condition (`OperatorAlone` or `OperatorNotAlone`), this is not interpreted in Fig. 3(b) in order to simplify type `A` as much as possible and to avoid additional arcs.

Events. Fig. 3 includes two types of event: one internal event (place `AssistenceCall`) and two external ones (`EmergencyCall` and `BrigadeTookResponsibility` places). Irrespective of the type of event, these are explicit signals translated into extra places receiving tokens from internal actions (`AskAssistence`) or the environment (`Generate`). These places send tokens to subactivities/actions (transitions) that require them. Thus, the net firing rule ensures that the transition receiving tokens cannot occur before the event has happened.

6.2 Validation and Property Verification

CPNs allow us to validate and evaluate the usability of a system by performing automatic and/or guided executions. These simulation techniques can also carry out performance analysis by calculating transaction throughputs, etc. Moreover, by applying other analysis techniques it is possible to verify static and dynamic properties in order to provide the complement to the simulation. Some of these properties are that:

– There are no activities in the system that cannot be realised (dead transitions). If initially dead transitions exist, then the system was bad designed.
– It is always possible to return to a subactivity if we wish (liveness). For instance, this might allow us to rectify previous mistakes.
– It is always possible to return to a state before (home properties). For instance, to compare the results of applying different strategies to solve the same problem.
– The system may stop before completion (deadlock). Thus, a work might never be finished, or it might be necessary to allocate more human resources to perform it.
– Certain tokens are never destroyed (conservation). Hence, resources are maintained in the system. This should be true for actors.

7 Conclusions

UML, with the nine notations embodied, is suggested as a general and standard notation for the analysis, design and development of object-oriented software systems. UML semantics is intended for the latter field; it is not a formal description. Neither does it define a standard process for its application, of which we are taking advantage, but is intended to be useful with iterative development processes.

In practice, the modelling power of UML is demonstrated by applying it to diverse types of systems. New approaches to address the analysis and design of cooperative systems must advance towards how to study and obtain interesting properties for systems. We argue that concepts should be correctly represented, at appropriate levels and that clear semantic links between them should be provided for useful integration, thereby resulting in a more powerful, useful and flexible system from all points of view.

Our main aim is to obtain the benefits of using an standard notation as UML:

– it allows us to model open, reactive systems [4], i.e., the main features of cooperative systems,
– there exist several UML-based tools to design and generate software (RationalRose, ArgoUML, ...), and
– new UML-based approaches are arising to model user interfaces [24,28].

On the other hand, it is necessary the application of formal methods to the engineering of cooperative systems in order to build CSCW systems correctly. For this purpose, CPNs have a graphical representation and well-defined semantics, which allows compact and manageable representations and therefore one more powerful analysis than that of UML.

References

1. Artman, H., Waern, Y.: Distributed Cognition in an Emergency Co-ordination Center. Cognition, Technology and Work, 1 (1999) 237–246
2. Dix, A.: "Formal Methods for Interactive Systems". Academic Press (1991)
3. Ehrlich, K.: Designing Groupware Applications: A Work-Centered Design Approach. In: Beaudouin-Lafon, M. (ed.): Computer Supported Cooperative Work. Wiley (1999) 1–28
4. Eshuis, R., Wieringa, R.: A Comparison of Petri Net and Activity Diagram Variants. In: Weber, Ehrig, Reisig (eds.): Proc. 2nd. International Colloquium on Petri Net Technologies for Modelling Communication Based Systems (September 2001) 93–104
5. Eshuis, R., Wieringa, R.: An Execution Algorithm for UML Activity Graphs. In Proc. UML'2001. LNCS 2185, Springer (October 2001)
6. Ellis, C.A., Gibbs, S.J., & Rein, G.L.: Groupware: some issues and experiences. Communications of the ACM, Vol. 34, No. 1 (January 1991) 38-58
7. Garrido, J.L., Gea, M., Gutiérrez, F.L., Padilla, N.: Designing Cooperative Systems for Human Collaboration. In: Dieng, R., Giboin, A., Karsenty, L., De Michelis, G. (eds.): Designing Cooperative Systems - The Use of Theories and Models. IOS Press-Ohmsha (2000) 399–412
8. Garrido, J.L., Gea, M.: Modelling Dynamic Group Behaviours. In: Johnson, C. (ed.): Interactive Systems – Design, Specification and Verification. LNCS 2220. Springer (2001) 128–143
9. Garrido, J.L., Gea, M., Padilla, N., Cañas, J.J., Waern, Y.: AMENITIES: Modelado de Entornos Cooperativos. In: Aedo, I., Díaz, P., Fernández, C. (eds.): Actas del III Congreso Internacional Interacción Persona-Ordenador 2002 (Interacción'02), Madrid, Spain (Mayo 2002) 97–104
10. Gehrke, T., Goltz, U., Wehrheim, H.: The dynamic models of UML: Toward a semantics and its application in the development process. Hildesheimer Informatik-Bericht 11/98, Institut fur Informatik, Universitat Hildesheimer (1998)
11. Gogolla, M., Presicce, F.P.: State Diagrams in UML: A Formal Semantics using Graph Transformations. In Proceeding ICSE'98 - Workshop on Precise Semantics of Modeling Techniques (PSMT'98) 55–72
12. Grudin, J.: Groupware and Cooperative Work: Problems and Prospects. Reprinted in Baecker, R.M. (ed.) Readings in Groupware and Computer Supported Cooperative Work, San Mateo, CA, Morgan Kaufman Publishers (1993) 97–105
13. Harrison, M., Thimbleby, H. (eds.): Formal Methods in Human-Computer Interaction. Cambridge University Press (1990)
14. Jensen, K.: Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use. Second Edition Springer(1996)
15. Jordan, B.: Ethnographic Workplace Studies and CSCW. In: Shapiro, D., Tauber, M.J., Traunmueller, R. (eds.): The Design of Computer Supported Cooperative Work and Groupware System. North-Holland, Amsterdam (1996) 17–42
16. King, P., Pooley, R.: Using UML to Derive Stochastic Petri Net Models. In N. Davies and J. Bradley, editors. UKPEW '99, Proceedings of the Fifteenth UK Performance Engineering Workshop, Department of Computer Science, The University of Bristol (July 1999) 45–56.
17. Malone, T.W., Crowston, K.: What is Coordination Theory and How Can It Help Design Cooperative Work Systems. Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'90). ACM Press, New York (1990) 357–370

18. McGrath, J.: Time, Interaction and Performance: a theory of groups. In Readings in Groupware and Computer-Supported Cooperative Work. R. Baecker (ed). Morgan Kauffman (1993)
19. Nardi, B. (ed): Context and Consciousness: Activity Theory and Human Computer Interaction. MIT Press, Cambridge MA (1995)
20. OMG: Unified Modelling Language Specification. http://www.omg.org (September 2001)
21. Palanque, P., Bastide, R.: Synergistic modelling of task, users and systems using formal specification techniques. Interacting with Computers 9 (1997) 129–153
22. Paternò, F.: Model-based Design and Evaluation of Interactive Applications. Springer-Verlag (2000)
23. Pettit, R.G., Gomaa, H.: Validation of Dynamic Behavior in UML Using Colored Petri Nets. UML'2000 WORKSHOP. Dynamic Behaviour in UML Models: Semantic Questions. On Line Proceedings (2000) http://www.disi.unige.it/person/ReggioG/UMLWORKSHOP/PROGRAM.html
24. Pinheiro da Silva, P., Paton, N.W.: User Interface Modelling with UML. In Information Modelling and Knowledge Bases XII. 10th European-Japanese Conference on Infomation Modelling and Knowledge Representation. Saariselka, Finland (May 2000). Kangassalo, H., Joakkola, H., Kawaguchi, E. (Eds.) Amsterdam, IOS Press (2001) 203–217
25. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language – Reference Manual. Addison-Wesley (1999)
26. Saldhana, J.A., Shatz, S.M.: UML to Object Petri Net Models: An approach for Modeling and Analysis. In Procceding of Twelfth International Conference on Software Engineering and Knowledge Engineering (SEKE2000)
27. Terveen, L.G.: An Overview of Human-Computer Collaboration. In Knolowledge-Based Systems Journal, Special Issue on Human-Computer Collaboration (1995) 67–81
28. TUPIS'00: Towards a UML Profile for Interactive Systems Development. Workshop of UML'2000. http://math.uma.pt/tupis00/
29. van der Veer, G.C., van Welie, M.: Task Based Groupware Design: Putting theory into practice. In Proc. of Symposium on Designing Interactive Systems (DIS'2000) New York (August 2000) 326–337