A Petri net-based support for object-oriented specification of complex control systems

Alioune Diagne, Jean-Michel Ilié,
Université Paris 6, Université Paris 5,
LIP6 - SRC,
4, Place Jussieu,
F-75252 Paris Cedex 05,France,
Tel.: (+33) (0)1 44 27 73 65,
email: {diagne,ilie}@lip6.fr

Daniel Moldt
Universität Hamburg,
Fachbereich Informatik,
Vogt-Kölln-Str. 30,
22527 Hamburg,Deutschland,
Tel.: (+49) (0)40 - 5494 2247,
email: moldt@informatik.uni-hamburg.de

Abstract

Object-oriented methodologies offer a set of concepts and means which allow to model complex control systems. For such systems, the designers need a methodical support like UML to handle the complex models and to coordinate the different activities that are performed along the life-cycle. For each of these activities, it is also important to have formal support in order to enhance quality. The dependability of such systems enforces the need to formally validate and verify their models.

In this paper, we address a Petri net-based formal support for object-oriented specification of complex control systems. Three complementary approaches, each of them providing a formal support for a given set of activities within the life-cycle, are combined in an overall one which concentrates on control aspects.

Keywords: Colored Petri Nets, Distributed Object-Oriented Control, Modeling, Prototyping, Validation, Verification

1 Introduction

The design of complex control systems puts high requirements on the modeling techniques. It requires advanced software engineering means to structure the systems in a way that takes profit from the widely used concepts like object orientation and methodologies as well as from constraints enforced by the application domains. The dependability of such systems enforces the need to ensure quality by validation and verification means along their life-cycle. In this paper, we propose and discuss the integration of three complementary approaches in an overall one that aims to cope with the previous needs. These approaches are based on use of object-oriented concepts and Petri nets to handle the activities of building models and their analysis towards quality enhancement.

State-of-the-Art

There are many different approaches for system specification varying from completely informal ones to completely formal ones. Within the area of object-oriented approaches, some are quite successful for information systems. Good examples for this are OMT, Booch or UML (see [RBP+91], [Boo93] and [BR97]). Beside several advantages, they provide standard software engineering concepts like abstraction and encapsulation. UML and other approaches can be seen as a well tested framework for the specification of systems. They fit well into new development environments by using the same concepts as they are used e.g. for programming. However, there is a lack of verification and validation techniques for the different models. This is especially true for the control view. Complex control systems often have to cover distributed and concurrent subsystems. Normal automata like finite state machines cannot be used sufficiently to describe the relevant aspects. Newly added features to UML like Concurrent Object Interaction Diagrams lack a formal semantics and are also restricted concerning the critical concepts mentioned above.

On another hand pure, formal approaches like algebraic specification (see e.g. [BFG+93]) are difficult to understand and apply, beside lacking typical software engineering practice. Among the formal approaches, Petri nets have specific characteristics like graphical representation, (direct) executabilities, powerful expressiveness etc. Nets allow partial or complete verification and validation of properties depending on the size and the structure of the system model.

Beside that, Petri nets have been integrated into the context of commercial approaches like structured analysis, Statecharts, or object-oriented analysis (see [Mol96]). This is a precondition for merging the nets with object-oriented approaches.

Already minor restrictions concerning the number of concepts within a technique allow to automatically derive from its outcomes Petri net models. These net models can then be analyzed concerning properties by verification and validation means. The aspect of taking into account constraints induced by architecture requires and allows different approaches. The pure analysis of systems which removes any design consideration is often too restricted in practice. Certain parts of a system exist and are not going to change. Good examples for this are computer networks or production systems (see [BM93]) where a great amount of investments have been made before and changes by introducing new features or subsystems do not allow to rebuild completely the environment of the planned system.

Overview of concepts and notions

The person who performs the process of software production is called software engineer. This process could also be performed by a group of people. The main distinction for people is related to their qualification. Users and software engineers are the main classifications. A specific person can have both roles. This person, however, must be aware of the separation between roles and which one he/she is playing at a given moment.

According to [Mol96], the software engineer is working in a certain *context* when applying an *approach*. The context is also called *environment* and provides certain means. An approach is based on *principles* and *paradigms* constituting its frame. Here the object-oriented ones are used. Additionally concurrency and distribution are important concepts which can be handled. Traditional approaches generally do not provide sufficient means for that.

Within the frame, the *facets* of an approach can be identified. Due to the restriction of the kernel of the proceeding only five facets are considered here: *method*, which is the way to proceed, *technique*, which is the mean to build models, *tool*, which is the mean to support all activities within an approach, *application*, which is the domain to be handled, and *resource*, which are the concrete limitations for a project.

The vision is to provide a proper set of concepts which give a sound foundation of an approach as sketched below. In general, UML techniques and Petri nets are used. The nets can be used instead the UML techniques or just as a support in the background for the verification. The usual approach of UML can be enhanced by executing its models. The modeler or software engineer gets a well tuned set of tools which allow to perform validation and verification very early. When looking at the advantages of prototyping (see [HI88]) the use of Petri nets is obvious. However, for prototyping the models need to be described in a precise way for execution. This execution can be either direct simulation of the models or the execution of generated code. Also the use of animation tools is very appropriate for the validation of models.

What is important to notice here, is that between the validation and the verification of these models normally there is a gap. By using nets this gap becomes much smaller.

Contribution

The approach proposed in this paper is an integration built from three complementary ones. The first approach concentrates on providing executable object-oriented models in general e.g. class diagrams (see [Mol96]). By using nets the formerly non-executable models become executable. This allows to "have a look" at the behavior of the specification meaning to express and validate it. However, due to the unrestricted modeling abilities and the resulting complexity of the models there are only few formal verification techniques available.

The second one focuses on the existing control model of UML (see [Dia97]). This allows to automatically generate a model to evaluate the control behavior. Typical questions answered with nets are deadlock freeness, liveness, or fairness. These can be verified without additional input by the modeler. However, to prove certain domain dependent properties some relevant information has to be added to the normal UML models e.g. by using specific pattern and supplying specific operating scenarios to check.

The third one provides an object-oriented approach which considers the underlying architecture of communication systems to be designed (see [HIT+97]). By using such architecture symmetrical parts of a system can be enhanced in order to ease both design and verification processes. Currently the approach is applied to protocols and distributed systems.

The results that will be presented in the remainder of the paper are:

- Static models like class diagrams become executable and therefore support validation in many different ways.
- The dynamic view of object-oriented specifications can – to a large extend – be verified concerning control properties.
- A smooth transition from the specification to design is presented and control properties of specification and design issues can be verified.
- A homogeneous basis for the different views within object-oriented approaches can be presented. Actually, in the approaches presented hereby, nets can stand for such a basis.
- A smooth transition between modeling, prototyping, validation, and verification can be found in a homogeneous approach.

2 The different levels of formal support

In this section three different levels of formal support are presented, discussed, and compared. The integration is done in the next section. The sequence is related to the different levels of preciseness and details required for each approach.

2.1 Formal semantics and validation

Especially for new areas and those which have never been investigated in the past, the modeler wants to get a *feeling* about the system. The same is true when a new project is started. The participants want to get a first idea about which parts of the planned system are most relevant to them.

This can be supported by building executable models or even prototypes. During the progress of the project, the interest of the participants become more and more precise. For this, more elaborated and tuned prototypes are of interest. Nets can be used as a basis or directly for prototypes due to their operational semantics. Furthermore, nets provide a basis for verification of properties on such prototypes, which will be covered in the sections 2.2 and 2.3.

The main activities of validation oriented activities are the building of the model, the animation, the simulation, the code generation, and the analysis. The basis is some kind of repository.

The usual techniques of object-oriented approaches allow to build models for specific views. These models usually cannot be executed. Nets can provide the means to execute them anyway.

During the project life-cycle more intuitive and executable means are required. Object-oriented approaches have their strength in the restriction to specific views based on semi-formal techniques. In [Mol96] and some related works, it is shown how to replace techniques normally used in specification approaches by nets or to provide a formal semantics for them. This was done for Data Flow

Diagrams, Control Flow Diagrams, Statecharts, Entity-Relationship Diagrams and Class Diagrams. There are only small restrictions necessary for each technique. However, the wide variety of different interpretations is therefore no longer possible. This results from the operational semantics provided by nets.

The operational semantics allows to execute the models. However, for this a transformation to nets is necessary. This transformation can be done automatically in the background for normal users. Only experts are working with the executable net models.

The restrictions on the techniques have proven to be not too strong, hence they conform to standard software engineering concepts. For instance for Statecharts global variables are no longer allowed[1]. These variables and their usage cause severe limitations concerning modularity. It would be easy to use a place for a variable. However, for larger systems, it does not seem to be reasonable to use global variables. Also the idea of a central clock (within all Statecharts of a model) had to be removed. By this, asynchronous communication becomes possible. This allows to spot problems within the specification and not to defer it to the design or even implementation level. If the design allows to use such a mechanism as a global clock, then nets with a modified firing rule could be used. However, the problems for the formal analysis have to be considered.

Furthermore, there is the possibility to integrate all different views on the basis of one single technique. Usually, this is done by a specific tool, the repository. Normally, this depends on the company which provides the tool. There is no general conceptual integration of the different techniques. This is overcome by the use of nets, hence only one formalism is used. The problem of finding redundant parts (via name and inscriptions) is still the same. This problem, which becomes quite dominant in large projects, is inherited by complex systems in general.

One major advantage is that also static models are transformed. One object can be considered to consist of several aggregates. These aggregates are themself objects. Each aggregate covers different views on this object (e.g. static, dynamic, functional). The aggregates can be refined, however. This is not considered here. This separation of concern allows a software engineer to concentrate on the relevant information. For the validation it is important to have many different possibilities in order to be flexible.

Executability of nets allows to perform efficient prototyping. The drawback is the lack of verfication. The different views can be transformed automatically from the semi-formal techniques to nets. To what extent software engineers work directly with nets or with the semi-formal object-oriented techniques shall not be discussed here, however, it is one of the choices that have to be made for a concrete project.

When moving to verfication the properties of interest are restricted to some views. Here in particular the dynamic view is covered.

Prototyping can be seen as a validation technique. However, it can also be seen as a way to produce final systems. This depends on the requirements concerning the efficiency of the implementation. For first versions, this is often not a central issue. The direct execution by simulators or the use of code generators is an efficient way for modelers to present the planned system to end users. These people get an impression on the behavior of the system.

One difficulty is that a normal simulator only provides output in net terms. Therefore it is necessary to provide animators. These animators allow to present the system in terms meaningful for the end user. However, this again implies some kind of workload on the modeler, hence the additional information concerning the application area and the user relevant data has to be include in the models somewhere.

[1]The use of global variables for the sake of simplifying the proof of properties should be allowed. One has to differentiate between the modeling of the application and the extensions for proofs.

2.2 Formal verification of control models

Control aspects in object-oriented specifications are split among many representations. Individual objects have methods and operations which model their internal control. This intra-object control can exhibit concurrency e.g. in presence of aggregation. State diagrams show the allowed intra-object control flow. The interactions between objects are represented by the control flow diagrams. They model the inter-objects control. The inter-objects control represents many aspects like co-ordination, communication and synchronization. Scenarios or event traces can be given to characterize some operating cycles of a system or their traces on some objects. All these aspects are given in different models but they are strongly related. For instance a scenario must be verified against the control flow models. On another hand, intra and inter-object controls must be be consistent versus each other.

2.2.1 From objects to nets: Guidelines and goals

The verification of control aspects relies on the level of preciseness with which they are expressed. The possibilities one has to express properties depend on the level of detail in the model of the control. The more information is contained in the control model, the more properties can be handled. The representation of the control is built from the dynamic and functional model. Activities and actions referred to in the dynamical model should therefore be matched by methods or be described in the functional model. Methods should be represented in a way that point out their control flow.

Actually, the persistent side effects of operation or activity execution can make some other invocations infeasible and then the control flow of an object can be no longer valid.

Another facility to be considered is to use somewhat simpler data representation to support the expression of the control. At the level of specification, data representation can be made of basic types like integers, characters and finite lists. This allows to avoid infinite state spaces for models. Also, it minimizes the tradeoff between complexity of tokens in nets and application of verification techniques. The more complex tokens are, the less verification techniques are available.

One modular colored net is synthesized from each object. In this modular net model, we model attributes by places and actions by transitions (see [Hei92]). Sequencing in activities and operations are represented by special places which constraint the firing of transitions modeling elementary actions.

The operational semantics of Petri nets allow to represent both the dynamic and functional models of commercial object-oriented approaches. Once modeled using nets, the corresponding diagrams can be validated by simulation or even formally verified. Simulation stands for model execution and can help to tune specifications involving end-users of a target system or its owners. It is based on the direct executability of nets.

2.2.2 Control properties: Representation and verification

Basic questions which can be answered using nets are liveness, deadlock-freeness and conservation of values beared by attributes. These questions can be answered by classical techniques in nets, like invariants computation (places and transitions) or model checking based on reachability graph construction. These properties can be considered as implicit expectations that are always worth to be verified on a specification because they constrain its correctness. Scenarios and event traces can be verified with more elaborated techniques. These can be considered as explicit expectations the modeler has to ensure on a specification. The construction of the reachability graph is achieved using modular techniques in order to hide non-relevant details, like the firing of autonomous actions in an object (see [Val94, MN94, CP92]). These techniques allow to stress the most important aspects or parts of a model and by the way allow to avoid the state explosion inherent to net analysis.

The representation of the control used to verify the properties is built mainly form the statechars with eventual refinements from data flows diagrams and the functional models. These refinements depend on the aspects one wants to stress on. For instance, to verify properties of an intra-object control model, a full detail representation of its control is needed. Therefore, the net built from the

Statecharts is refined by the information from the functional model. The data flow diagrams are used when one wants to stress on the inter-objects control (i.e. on the interactions). Only actions observable for the environment of an object need to be represented and its internal details can be hidden.

The two levels of preciseness are strongly dependent. The intra-object control needs to be verified before the inter-objects one, because it supports it. The basic properties which are considered first are safety and reliability properties. A reachability graph is built for an object assuming that interactions occur correctly. On this graph, terminal states (i.e. states without successors) must be checked as being deadlocked ones. For each deadlocking state, we can give the sequence of actions leading to it with the traceability information. Livelocks and repetitive transition series can also be derived from this graph. Once this intra-object control of an object is thoroughly verified, we can consider that its dynamic model is safe. The reliability is shown using event traces or scenarios on aggregations built from sets of interacting objects. Nets synthetized from Statecharts can be combined by fusion of transitions modeling shared events.

2.3 Design aspects and formal verification

Some of the standard formal verification approaches are based on the analysis of reachability graph representing the system states and their changes. Due to the state explosion, challenges appear in order to reduce the representation of reachability graph while improving span algorithms used in verification processes. Original researches have shown the interest of exploiting the intrinsic symmetries that exists between some objects within the system (see [Jen92]). The existence of such symmetries causes redundancies of information in the reachability graph, that means sub-graphs having the same structure and expressing similar behaviors. Thus, the verification process can be restricted in order to only analyze in the graph structures which have no redundancies. Moreover, general properties can be automatically checked on such graphs with respect to a convenient (but static) analysis of the specified property (see [IA97]). Conversely to standard approaches of verification, symbolic questions can be asked in order to test the behavior of any element of a group of objects that behave symmetrically. From an object-oriented view, the ability of declaring classes and using the associated mechanism of instantiation does not necessarily ensure that two objects of the same class have the same dynamic behaviors for the system. Researches have shown the necessity to detect groups of objects having the same capabilities of communication [ZHM93]. This can be achieved by inheritance and refinement in order to define classes whose instances have the same dynamic capabilities. It is another grouping level inside classes which allows to define these refined sub-classes. Such an approach is general, but has a straightforward interest in application domains like protocols for which the communication between entities are supported by an explicit structure, like a ring.

Based on that principles, an object-oriented model has been proposed, namely HOOSL (see [Aja95, HIT+97]). A specification in HOOSL is currently composed of three parts :

- the architectural part of HOOSL describes the potential interactions between sub-systems. Such sub-systems are generic in the sense that they can be instantiated several times. The description is given hierarchically to enable the reuse of these generic descriptions. In this part, topological links between generic entities specify how the represented objects are connected (e.g. ring, multi-point connection) and the discipline under which they communicate (e.g. FIFO, strong synchronization). Like in Estelle, links between sub-systems of the same level can be used through the hierarchy in order to specify the connection of sub-systems of distinct levels. Currently, attribute types are willingly limited to keep a well-formed nets semantics, thus allowing optimized verification on this basis. A reuse of a link through the hierarchy can be viewed as an inheritance of links.
- the behavioral part of HOOSL specifies the behavior of each class. Specializations can be used to avoid redundancy between similar class descriptions.

- the last part of HOOSL concerns the instantiation of the system for which verification is required. Instantiations are processed in a bottom-up approach with respect to the hierarchy defined within the architectural part. This bottom-up approach approach allows to define the instantiation of sub-systems of the architecture that can be re-used several times (and sometimes differently) in the instantiation of the upper level part. Therefore, such a process is more an instantiation of architecture than instantiation of classes of objects. Conversely to standard formal description technique like Estelle or LOTOS, HOOSL allows the reuse of generic descriptions while keeping the notion of symmetry between groups of objects. Therefore, a symbolic graph can be computed automatically from such specification and a temporal logic property, in order to perform efficient verification.

2.4 The three approaches are indeed complementary

The first approach supports the execution of formerly semi-formal models and provides a formal semantics for them. Therefore, it allows only one interpretation of a given model and provides a basis for validation by net execution. However, the approach lacks verification of its models. The main reason for this is that the models are not restricted strongly enough to match the requirements of todays formal verification analysis tools.

This problem is addressed by the next two approaches. They restrict the nets in such a way that formal verification can be applied. Concentrating on the well handled area of control aspects in models, one can prove, even for complex models, several properties on the dynamic view on a system. The second approach suffers from the size of the model. For huge models, the class diagrams can be refined in order to enforce the exhibition of symmetries so that the third approach remains applicable. The drawback is that the more application dependent properties are under investigation, the more the models have to conform to the kind of nets that can be handled nowadays. This drawback however, is a principle problem of complex systems. The third approach tries to cope with the state-explosion problem in a way that uses inherent symmetries of the systems. Due to precise proofs, design issues are taken into account, when applying the third approach. Even if it starts also at an high level of abstraction, more concrete facts of the design and underlying architecture are used. This helps to refine the class diagrams in order to exhibit the symmetries.

The first approach handle the modeling activities while the two other ones work on the issued models. Results from verification / validation can be used to adjust the model accordingly. The verification / validation aspects allow to evaluate the model produced by the means of each of the three approaches. These approaches and their integration is complementary to object-oriented ones in the sense that they enhance it qualitatively.

3 Integration of the different approaches

In this section, the integration of the former approaches within one overall approach is shown. This is done by sketching the way a modeler has to proceed when building a system. It is important to notice that there is not only one optimal approach. Depending on many different aspects each project has to use a special (set of) approach(es). Nevertheless, there are some common features and the focus hereafter will be on them. In the next section the general assumptions are shown. Then different parts of the proceeding are explained in more detail. Finally features that are not common in all approaches are shortly discussed or just sketched or mentioned.

3.1 Assumptions and goals of the overall approach

Existing object-oriented approaches can be enhanced by prototyping in general and more specific by formal verification of the dynamic model. However, in order to use a set of concepts and notations which are well known and widely accepted, UML is chosen here. This allows to have a smooth transition from traditional approaches to the new one here. Also from the perspective of net modelers, now there is an approach to derive the specification in a non pure intuitive way.

3.2 How to achieve the integration?

A system designer starts by building a model for his/her target system. The building of the model can start from scratch or use any other strategy like reuse or modification of existing ones. A combination of these strategies is also one possible way of building models. End users and system owners are involved in the building of the models. The models built are stored in a repository which acts as a resource to coordinate the different activities along the life-cycle. Based on the transformations into nets and their direct executability, models of systems can be:

- simulated to detect misconception or to check completeness and global coherence of the model.
- animated to check their correctness against the initial expectations.

The validity of the model is the starting point for its verification. The purpose is now to check the properties expected on the target solution. All these activities can raise errors which must be corrected by modifying the models. The part of the process described until now is an iterative one. Once a satisfactory models obtained, we generate code from it. Code generation minimizes the eventual derive between the model and its implementation. It also enhances the prototyping approach we use here.

The integration of the different approaches presented previously into an overall approach is now described. When building the models, the system designer begins to sketch the architecture. He/she describes therefore the main objects (or classes) and adds relationships between them when one comes along. The first approach gives the necessary means to handle such models. They can be simulated, animated and by this support its analysis in terms of completeness and global coherence. This first models are afterwards refined by going into details of control. The system designer goes on performing object design by describing the control that governs the behavior of each object and the way control flows from object to object. The second approach gives the means to verify properties expected on this detailed specification of the control. Simulation and animation can be performed at different levels of abstraction e.g. by stressing on interactions between objects or by looking at their internal details. The design can be refined by adding information in order to enhance a decomposition into sub-systems with symmetries between their components. In the case that the architectural aspects become important or the nets contain symmetries, the third approach is applied.

Now one possible scenario is described for a project:

1. initialize repository
2. build or modify informal models (static, dynamic, functional)
3. discuss with users (eventually execute models)
4. iterate the last two subtasks
5. enhance model for executability
   (a) simulate and debug
   (b) animate
   (c) analyze
   (d) generate code
   (e) execute/use produced system (eventually direct use of the simulated model)
6. use new experience to enhance models
7. add new aspects (qualitative or quantitative aspects)
8. repeat this validation process until it is appropriate to verify some relevant properties
   (a) transform class diagrams / Statecharts into nets
   (b) expand the model

(c) compute the reachability graph / symbolic reachability graph

(d) verify the properties

(e) modify the models according to the results

9. validate reached models
10. finish or modify the models and therefore start again

This scenario is also a framework. However, it also reduces the possible actions of the system engineer. The intention and the general goals influence the steps taken considerably. The emphasis can be put on any of the tasks mentioned.

Reasonable aspects to be different between each project/scenario:

- number and kind of views or semiformal models used
- quantity / quality of the process and product (specifications)
- goals that should be achieved by the process
- total stop of the project process
- number of involved people (different environment concerning project management)
- relevance of results for company (ranging from uncritical test to dead-or-life projects)
- tool support (especially the functionality of the repository (ranging from a simple file system to an integrated software development environment))
- safety critical application (resulting in more properties to be verified)
- ...

3.3 Discussion of integration

One of the main difficulties is to provide the additional information according to UML, this is done by just adding some formerly not used inscriptions and following some patterns when building the models. The other problem is to use the results of the verification to improve the model. For this, the parts of the transformed UML models (which are the executed Petri nets) have to carry the information about their origin. Hence the verification is performed on unfolded well formed Petri nets (see [CDFS90]), this can only be done with tools. These tools provide the right information to the user. This is done by overloading the nets generated from UML models with traceability information which allow the feedback of results from verification and validation. This information realizes a correspondence between net entities and UML entities.

This reverse direction is usually quite difficult. However, it becomes quite obvious once we have specified exactly what parts have to be transformed from the UML model into a Petri nets component. This allows to solve the problems due to the overlapping of the models. The relevant parts have to be recognized and matched accordingly.

If design issues have to be included, then it gets even more complicated in the sense that the model cannot be longer considered to be a specification. At the same time, it depends again on the intention of the modeler if the direct extension of the specification is reasonable.

It is important to notice at this point, that there is no gap between the specification and the design of a system. This is due to the use of the same technique. In [RBP+91] the transition from the specification to the design is rather fussy.

The traceability of proved properties through refinement is not yet possible in the overall approach. It is being investigated but the main difficulty is to make it transparent for the refinement processes which are allowed in methodologies like UML. Enforcing traceability would result in constraints on such methodologies which are not acceptable for common software production processes.

Formal semantics and validation of models

The validation of UML models relies heavily on their executability. Only reading is not appropriate because the dynamic behavior is not visible. In [Mol96] for object-oriented concepts an operational semantics in form of Coloured Petri nets by Jensen (see [Jen92]) is given. This is also applicable for the techniques used in UML.

Initially the validation approach relies, as described above, more on the modeling. Later the execution becomes more and more important. When combined with the other approaches, a strong interaction between the verification and the validation becomes established, co-ordinated by the modeling activity.

The results of the modeling can be seen as prototypes. For the validation itself, it depends on the execution. Animation and background simulation can be done without nets. However, if the behavior has to be observed directly, then nets are more appropriate. Then a testing and debugging method can be used.

Results of the validation are used directly by the software engineer to modify the models.

Specification Verification Approach

The verification of control aspects in object-oriented models needs first an integration of the different representations. The starting point is the Statecharts modeling the dynamic of objects. A refinement step is undergone with the knowledge on the system supplied through the other representations which carry control information (control flow and data flow diagrams). This causes the nets we synthesize to be huge and to present an overall view on the system. All verifications are run in the net environment where we can build some temporary aggregates which are not relevant for the object-oriented methodology.

The implicit properties allow to evaluate the control aspects for each class of objects. It is some kind of corroboration of the level of confidence attained with the first method (see 2.1). At this first level of verification, we assume for each class (or object) that its environment (others classes or objects) is safe and reliable. Results of verification are easily traced backward to the UML models and they allow to refine the modeling activity.

Once this first level is achieved, the system modeler can undergo verification of explicit properties which involve interactions between objects. For this verification, aggregations are built in the net environment and the results are fed back according to the traceability information which realizes the correspondence between net entities and object-oriented ones. Once again, the results can help to refine the models by rerunning modeling activities.

The refinements resulting from verification results are expected to happen in the object-oriented methodology according to interpretation of these results by the system modeler, his/her intention and ability to rebuild or modify the model. This is especially true for false explicit properties when they involve reused parts which cannot be modified or dropped away.

Design Verification Approach

In the presence of objects running the same dynamic behavior, the approach presented in 2.3 can be used in UML context by offering symbolic means to express interactions between classes. With well-formed Petri nets semantics (see [CDFS90]), this allows to supply optimized verifications and brings out the interest of integrating the HOOSL approach within UML methodology. The major requirement is to refer to classes systematically instead of singular objects. Different views of UML are inputs for such integrations:

- Classes to be considered are defined by the class diagrams.
- Descriptions like Statecharts are used to define the behavior of classes.
- Control flow diagrams refines the potential interactions between classes.

The outcome of HOOSL consists in a given instantiation of the system in the sense that it helps the software engineers to determine groups of objects running similar behaviors (thus offering the same services), and to define generic relations between them. In an object-oriented approach, the definition of such groups is captured in terms of classes by means of inheritance mechanism applied on original classes: it consists of a finer grouping of objects such that generic relations between them are identified. Hence, HOOSL induces refinements on class diagrams and control flows. Verification is then allowed for a system with an *a priori* known maximum bound number of instances for each class. Moreover, two kinds of traces can be obtained for validation purpose:

- Symbolic traces which show at each step (sub)groups of objects able to react to given events.
- Unfolded traces which show the behavior of each individual object.

Finally, prototyping is enabled through an implementation of classes with a choice between two kinds of solutions: (1) implement explicitly generic class links as methods to be invoked or (2) instantiate the generic relations on the basis of object identities.

The overall approach as a process which integrates the advantages of each contributing approach has to be considered over time with respect to the different tasks in a project. The workload for a project is composed of the main following tasks:

- building and modifying models while enroling the user in order to always stick to the target system,
- validating the models,
- verifying properties on those models whence they are considered valid.

As long as one proceeds the workload of modeling and validation should decrease while the one of verification should go up. Modeling should stop when the verification succeeds. Validation should last afterwards to check possible modifications on the model due to results from the verification. The correctness of a model should be asserted against its validity towards the expectations of the end-users.

4 Conclusions

The approach presented in this paper allows to enhance the methodological support supplied by the UML with validation and verification capabilities. Analysis and design of complex control systems can thus be considered within UML with an increased degree of formality. Many of the aspects of control in such systems (co-ordination, communication and synchronization) can be handled with nets. Futhermore, the static models of a system are made executable here. Therefore, misconceptions and errors affecting the coherence and completeness of the models can be detected and corrected earlier. If such errors are corrected later in the life cycle, the costs are much higher.

Adopting existing approaches and then integrating them leads to significant improvements. Software enginnering activities are therefore qualitatively enhanced. This results in some extra-work needed from the software designers to tune their models to reach the level of preciseness necessary for the verification and validation activities. But this is an inescapable cost to be payed against the level of confidence on the models. Therefore, the balance between the advantages and the disadvantages is a principle problem. This problem has to be solved according to the needs of the verification and validation, and the ability to afford them.

The approach based on Petri nets presented here is one way to provide such a formal support for UML. It is an ongoing work and the focus will be put on the integration of the three approaches which have evolved independently before. This requires concepts and tools realize them. The most important ideas have been sketched here. They are: use of one technique namely Petri nets, prototyping in the sense of small steps while building models, neat integration of verification and validation based on Petri nets, combination and integration of a commercial available methodology with a formal technique, and executability of all UML models.

References

[Aja95] Khalil Ajami. Syntax and Semantics of Hierarchical Object-Oriented Specification Language HOOSL. DEA Report, in french, DEA MIASH of Paris V University, Paris V University, Sept 1995.

[BFG+93] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen. The requirements and design specification language spectrum – an informal introduction; version 1.0. Technischer Bericht TUM I9311-12, Technische Universität München, 1993.

[BM93] A. Borusan and D. Moldt. A Method for Developing CIM-Systems with Coloured Petri Nets. In Michel Cotsafis and Francois Vernadat, editors, *Advances in Factories of the Future, CIM and Robotics*, pages 91–100. Elsevier, Amsterdam London New York, 1993.

[Boo93] G. Booch. *Object-Oriented Design*. Benjamin/Cummings Redwood City, CA, 2nd edition, 1993.

[BR97] G. Booch and J. Rumbaugh. Unified Method for Object-Oriented Development. Teaching documentation set, Rational Software Corporation, Santa Clara, USA, 1997. Version 1.0.

[CDFS90] G. Chiola, C. Dutheillet, G. Franceschinis, and Haddad S. On Well-formed Coloured Nets and their Symbolic Reachability Graph. In *Application and Theory of Petri Nets 1990*, Lecture Notes in Computer Science. Springer-Verlag, June 1990.

[CP92] S. Chirstensen and L. Petrucci. Towards a Modular Analysis of Coloured Petri Nets. In *Proceedings of the International Conference on Application and Theory of Petri Nets*. Springer-verlag, June 1992. Sheffield, UK.

[Dia97] A. Diagne. *Une Approche Multi-Formalismes de Spécification de Systèmes Répartis: Transformations de Composants Modulaires en Réseaux de Petri*. Dissertation, University of Paris 6, LIP6, 4, Place Jussieu, 75252 Paris Cedex 05, May 1997.

[Hei92] M. Heiner. Petri nets based Software Validation, Prospects and Limitations. Technical Report R92-022, GMD/First at Berlin Technical University, Germany, March 1992.

[HI88] S. Hekmatpour and D. Ince. *Software Prototyping, Formal Methods and VDM*. Addison-Wesley Publishing Company Inc., Wokingham,, 1988.

[HIT+97] S. Haddad, J-M. Ilié, M. Taghelit, C. Dutheillet, and P. Moreaux. Contribution des réseaux de Petri de haut-niveau à la validation dún réseau de communication haut-débit. *(journal in french) revue RAIRO des Techniques des Sciences Informatiques*, 1997.

[IA97] J-M. Ilié and K. Ajami. Model Checking through Symbolic Reachability Graph. In . Proceedings of the CAAP9́7, April 1997.

[Jen92] K. Jensen. *Coloured Petri Nets: Volume 1; Basic Concepts, Analysis Methods and Practical Use*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin Heidelberg New York, 1992.

[MN94] T. Murata and M. Notomi. Hierarchical reachability graph of bounded nets for concurrent software analysis. *Transactions IEEE on Software Engineering*, 5(20):325–336, 1994.

[Mol96] D. Moldt. *Höhere Petrinetze als Grundlage für Systemspezifikationen*. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, August 1996.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premeralani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1991.

[Val94] A. Valmari. Compositional Analysis with Place-Bordered Sub-nets. In *Proceedings of the International Conference on Application and Theory of Petri Nets*. Springer-verlag, June 1994. Zaragoza, Spain.

[ZHM93] B. Zouari, S. Haddad, and Taghelit M. A Protocol Specification Language with a High-level Petri Net Semantics. In *Proceedings of the International Conference on Decentralized and Distributed Systems*. Springer-verlag, September 1993.