DESIGN OF DISTRIBUTED MANUFACTURING SYSTEMS USING UML AND PETRI NETS

B. Bordbar, L. Giacomini and D.J. Holding

Department of Electronic Engineering, School of Engineering, Aston University,
Aston Triangle, Birmingham B4 7ET, UK
Tel: Tel: +44 (0)121 359 3611 Fax: +44 (0)121 359 0156
e-mail: {B.Bordbar,L.Giacomini,D.J.Holding}@aston.ac.uk

Abstract: This paper describes the design of a supervisory control system for a distributed manufacturing process, which forms part of a wider manufacturing system. The focus of the paper is on the design of a verifiable discrete event controller using a UML based method. The approach adopted involves (i) using Petri net models instead of conventional Statecharts to provide analytic Dynamic Models; and (ii) using compositional Petri net techniques to synthesise the Interconnection Model. The model of the complete controller can be then analysed and verified using Petri net theory. The approach is demonstrated by application to a prototype packaging machine. *Copyright© 2000 IFAC*

Keywords: Discrete-event dynamic systems, Petri-nets, Object modelling techniques, Manufacturing systems

1. INTRODUCTION

Recent advances in computer technology have resulted in a widespread use of Discrete-Event Dynamic Systems or DEDSs in manufacturing, robotics, traffic management, logistics, and computer and communication networks (Cassandras, 1999). DEDSs require complex control systems (Ramadge and Wonham, 1987) to ensure correct and optimal operation. To model complex DEDSs, researchers have developed bottom up, top down and hybrid synthesis techniques. However, these approaches concentrate on functional abstraction, and have produced incomplete specifications and designs (Firesmith, 1993). In order to facilitate the design of complex systems, produce more understandable designs and specifications, facilitate the transition between design and implementation and to enable software re-use, several researchers including Booch *et al.* (1999), Douglass (1999), have advocated a paradigm shift towards object oriented (OO) techniques.

The Unified Modelling Language (UML), originally a methodology for software designers, is the most recent product generated by the aggregation of previous generation Object Oriented methodologies (Booch et al., 1999). UML takes the designer through the design life cycle, starting from the description provided by users or experts down to the final software product. It preserves convergence and clarity in design by prescribing a set of steps that generate an evolving model of the system, and facilitate the rigorous examination of this model. Thus, the application of UML by different people with different skills results in comparable and highly portable final designs.

UML consists in a set of nine main graphs or charts with explanatory comments that can be expressed in a formal way or in plainspoken language. These fall into two categories, static aspect diagrams and dynamic aspect diagrams, and the designer can choose quite freely to use a subset of them. In UML the dynamics of objects are described using a form of state diagram known as a Statechart (Harel, 1987). Concurrent or distributed systems are formed by creating parallel State Charts that are inter-connected and synchronised using *interaction diagrams*.

Although Statecharts are very popular and are well supported by implementation tools, they currently lack analytic capabilities and thus software tools cannot ensure the functional consistency of the overall design. Conventionally the behaviour of such designs is investigated using process considerations, such as completeness arguments (Levenson, 1995) and are demonstrated by simulation. However, to facilitate analysis, any system described by a State Chart can be replaced by an analytic representation such as Process Algebras, Automata, or Petri nets. Among the alternatives, Petri nets have a graphical approach that is easy to understand (Murata, 1989) and they are more effective in describing concurrent and asynchronous systems. Petri net theory can be used to analyse DEDS characteristics such as synchronisation, concurrency, conflicts, resource sharing, precedence relations, event sequences, non-determinism and system deadlocks (Desrochers and Al-Jaar, 1995). Also Petri nets, unlike state diagrams, are modular, and larger nets can be formed by simply merging places or transitions.

In this paper, to enhance analytic capabilities we shall improve our model by substituting the State Chart representation of dynamic models with a Petri-net. The paper also presents a method of synthesising coordination and synchronisation logic for distributed or large scale designs using Use Case information and compositional Petri net techniques. The approach is demonstrated by application to a manufacturing system comprising a prototype packaging machine.

2. UML BASED DESIGN

2.1 *Use Case and Class diagrams.*

The UML design procedure (Booch et al., 1999) starts with the study of the Use Cases which are detailed written descriptions of ‘what the objectives are’ and 'how the job is carried out'. Studying the use cases enables the designer to recognise different '*key agents*' of the system (*Objects* in UML terminology).

Considering common features and operations of key agents, objects are extrapolated into collections called *Classes*. Classes can be organised in a graph (or a collection of graphs), to build a '*class diagram*', that describes the *static relationship* between the classes. The classes are represented graphically by rectangular boxes accomodating lists of attributes and operations and are connected together by lines or links that can be either of association type or of generalisation type. An *association* is a structural relationship that specifies the connection between one or more members of the classes. A *generalisation* is a relationship between a general class and a derived class, i.e. one defined from another class by means of inheritance. The operations defined in the class diagram include all the services that can be requested from an object to effect the behaviour. For the manufacturing system applications we have in mind, the system can be arranged in such a way that all the synchronisation issues can be expressed in terms of Boolean attributes of the involved classes.

2.2 *Petri Net Dynamic Model.*

The dynamic model describes behavioural aspects of the object classes, in the sense that they describe the sequence of operations that occur without regard for what the operations do, what they operate on, or how they are implemented. To improve the representation and facilitate analysis of the UML dynamic model, in this paper the dynamic model is represented by a Petri net. For general information regarding Petri nets, we refer to Murata (1989). Generally, the Petri Net of a class is formed by using a place to represent each Boolean attribute and a transition for each operation that changes the attributes values. A token in a place means that the attribute value is set to true (false otherwise).

2.2 *Graph of Desirable States (GDS) and Compositional Petri Net.*

The process of compositional synthesis is not an ad-hoc procedure. Simply decomposing the Use Case diagrams into a bag of rules that are imposed on the objects ignores the important sequence information and will over constrain the model.

To maintain the precedence relationships and attain the synchronisation objectives specified in the Use Case we construct a directed graph, which we shall refer to as the *Graph of Desirable States* (GDS), which enumerates all desirable states and their relationships. The GDS maps the Use Case information into the Petri net domain, i.e. the sentences in the Use Case are translated in sets of rules in terms of places and transitions. The word “desirable” reflects the facts that the graph embraces all we expect the system to do, and any unwanted or undesirable behaviour is prohibited by identifying (for the design of constraint or inhibition logic) all enabled transitions that lead to undesirable behaviour.

2.3 *The Graph of Desirable States*

Let us assume that our system is made of m objects. For each object a Petri net is instantiated. Assume that Γ denotes the part of the Use Case dealing with the synchronisation of n of the above components into an overall system (typically, the objects are synchronised two at a time, until the compositional approach encompasses the whole system).

Assume that $(\boldsymbol{N}_1, \mathbf{m}^1{}_0), \ldots, (\boldsymbol{N}_n, \mathbf{m}^n{}_0)$, where $\mathbf{m}^i{}_0$, i=1, …, n, denote the initial markings, are bounded and live Petri Nets, representing object instances of these n components of the system. A proportion of the information provided by Γ has already been captured in the body of the dynamics of the Petri nets $(\boldsymbol{N}_1, \mathbf{m}^1{}_0), \ldots, (\boldsymbol{N}_n, \mathbf{m}^n{}_0)$. Let $R_\infty(\boldsymbol{N}_i, \mathbf{m}^i{}_0)$ denotes the set of all reachable markings of the Petri Net $(\boldsymbol{N}_i, \mathbf{m}^i{}_0)$. For each $\mathbf{m}^i \in R_\infty(\boldsymbol{N}_i, \mathbf{m}^i{}_0)$, let *enabled*$(\mathbf{m}^i)$ denote the set of all enabled transitions of $\boldsymbol{N}_i$ under the marking $\mathbf{m}^i$. Each node of GDS is labelled by a (n + 1)-tuple of the form $a = (\mathbf{m}^1, \ldots, \mathbf{m}^n, U)$ where $\mathbf{m}^1, \ldots, \mathbf{m}^n$ are reachable markings of the components $\boldsymbol{N}_1, \ldots, \boldsymbol{N}_n$ and U is the set (possibly empty) of undesirable enabled transitions under $\mathbf{m}^1, \ldots, \mathbf{m}^n$, as derived from the use case Γ. Thus U is a subset of of *enabled*$(\mathbf{m}^1) \cup \ldots \cup$ *enabled*$(\mathbf{m}^n)$. For the node labelled with $a = (\mathbf{m}^1, \ldots, \mathbf{m}^k, U)$ we shall write $\boldsymbol{m}(a) = (\mathbf{m}^1, \ldots, \mathbf{m}^k)$ and $U(a) = U$.

The GDS can be generated as follows. Consider the set E_0 of all transitions enabled under initial marking $\mathbf{m}^1{}_0, \ldots, \mathbf{m}^n{}_0$. From the above, a possibly empty subset U_0, of the transitions (or more properly their associated actions) are undesirable. Create the first node, which shall be referred to as the *initial node*, and label it with $a_0 = (\mathbf{m}^1{}_0, \ldots, \mathbf{m}^n{}_0, U_0)$. From this node start firing each of the desirable transitions $E_0 \backslash U_0$, to obtain another set of nodes each with their marking and a set of undesirable transitions. Put arcs connecting node a_0 and the newly created ones labelling them with t, where t is the name of the corresponding firing transition. The procedure is repeated for each of the new nodes created.
The GDS captures the behaviour expected from the composite net. For example, starting and ending in the same node of GDS represents a cyclic phases of the system. The GDS will also reveal problems with a design: for example, if there is a node *a* of GDS with no output then our design of the system expects a deadlock, which is anomalous.

Remark: The algorithm creates at most $\alpha^n 2^{n\beta}$ nodes, where α is the maximum number of reachable states of components and β is the maximum number of enabled transitions under different markings. Notice that each subset of the set of enabled transitions can be potentially a non-desirable set of transitions.

2.4 *Connecting the components Petri nets*

Consider the task of interconnecting together the Petri net dynamic models $(\boldsymbol{N}_1, \mathbf{m}^1{}_0), \ldots, (\boldsymbol{N}_k, \mathbf{m}^n{}_0)$ to create a composite Petri net $(\boldsymbol{N}, \mathbf{m}_0)$ with the desired coordination and synchronisation as described in the Use Case. The composition is performed using standard Petri net techniques (Juan et al., 1998) and the information in the GDS concerning desirable and undesirable transitions. For example, an almost general rule applicable when we want to prevent a transition t_k in Petri Net N_j from firing under a certain marking $\mathbf{m}^i{}_k$ in Petri Net N_i, a new place is added and connected as input/output to the transition t_k. The place is also connected to transitions in Petri net N_i in such a way that, when the transitions give rise to the marking $\mathbf{m}^i{}_k$, the token is removed. The firing of the transitions moving out of the marking $\mathbf{m}^i{}_k$ will put the token back in the place (see figure 1).

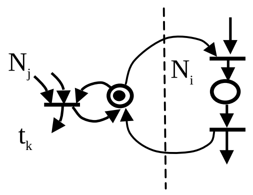

Fig. 1

Although, the composition process is straightforward for a GDS in which for $\forall$ *a, b* nodes, $\boldsymbol{m}(a) = \boldsymbol{m}(b) \Rightarrow U(a) = U(b)$, in other cases, significantly, the sequence information must be used to distinguish the states in the compositional process. When the composite Petri net is complete, it can be analysed using to Petri net theory to ensure that it is deadlock-free, live and bounded. The method is illustrated in the following example.

3 APPLICATION TO A PRODUCTION LINE PROCESS

The approach is demonstrated by considering the design of a controller for a simplified production line comprising loosely-coupled independently-driven mechanisms as shown in figure 2. The major components of the system are controlled individually and independently and perform motion profiles corresponding to different tasks. Supervisory (discrete event) control is to be used to synchronise the components.

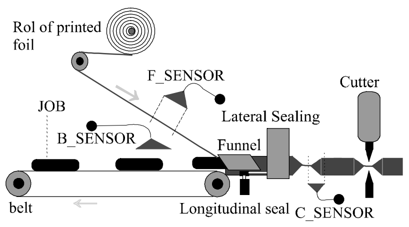

Fig. 2. Production Line.

The wrapping system of figure 2 is made of 4 objects: the belt, the foil roll unwinding device (film), the welder, the cutter. The product (JOB) and the foil that carries a printed tag (TAG) are identified with their supports, i.e. the belt and the film, respectively. JOB and TAG are displaced with respect to the belt and film. Let us examine the Belt Use Cases. When the JOB arrives (*new* JOB) in the proximity of a decision point sensor (**dp**), the state of the TAG is evaluated. If the TAG is at decision point the wrapping can take place (*go*). However, if the TAG is still **out**side the wrapping area, the JOB will stop, **wa**iting for the TAG to arrive at its decision point (*ab*ort operation). When the TAG arrives (*new* TAG), the JOB is restarted (*st*art leading to the **wr**apping state). When JOB and TAG are both in the **wr**apping state, the packaging foil is formed into a tube via a funnel, and a longitudinal sealing roller welds the two edges of the film together. The tube is sealed between packs by a lateral sealer (welder) and the wrapped product exits from the **wr**apping area (*ex*it leading to the state **out**). The sealed products are then separated by a cutting machine (cutter) to produce individually packaged products, and the whole cycle restarts.

Similar dynamic models have been derived for the Film (TAG), Welder and Cutter. The welder and film, and film and cutter, are synchronised by applying a heuristic similar to the one between the belt and film.

3.1 *The class diagram*

The description in Section 3 plays the role of the Use Case for the production line of figure 2. The underlined terms represent the classes: Film, Belt, the Welder, and Cutter. The product to be wrapped, JOB, is identified with the belt. The printed film and the motor driving the unwinding are also identified with the Film object. The terms in bold typeface are the attributes of the classes (for the Belt, B_dp, B_wait, B_wrap, B_out; F_dp and similarly for the class Film). The terms in italic typeface are the operations of the class. As an example, for the Belt:

B_new() {B_out=False; B_dp=True;}
B_ab() {B_dp=False; B_wait=True;}
B_go() {B_dp=False; B_wrap=True;}
B_go() {B_wait=False; B_wrap=True;}
B_exit() {B_wrap=False; B_out=True;}

The class diagram for the production line of figure 2 is shown in figure 3, without the attributes/operations lists.

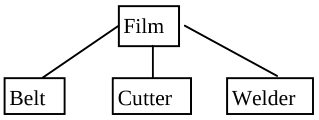

Fig. 3. Class diagram.

3.2 *Synthesising Petri net model*

For conciseness, we will focus on the interaction and synchronisation of the Belt and Film. First Petri nets are derived for each of the classes by assigning one place to each attribute and one transition to each operation. The Use Case description of the dynamics of each object is then used to construct the Petri net dynamic model. Specifically, places associated with attributes that an operation sets to False (or True) form inputs (or outputs) of the associated transition. In this particular application, the dynamic models of the classes are all structured in the same way as shown in figure 4 (a)-(b).
The initial marking for each instantiated object is obtained by considering the initial state of the corresponding components of the production line. The system starts with B_out and F_out.

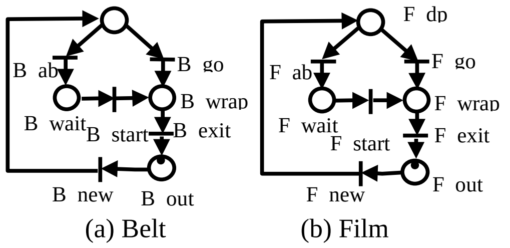

Fig. 4. Petri Net for the classes Sensor and Belt.

The GDS for the mutual synchronisation of Belt and Film. The next stage in the compositional process is designing the synchronisation logic that enforces the mutual synchronisation heuristic for the Belt and Film, as defined in Section 3. To do this, we make use of the discursive Use Case provided in Section 3, and the Petri Nets of the components, to generate the GDS, using the procedure described in Section 2.4.
For example, starting with (B_out, F_out) we will have a set of enabled transitions enabled: B_new and F_new. From the Use Case, since no JOB or TAG

are yet present, none of the two transitions is undesired, therefore U = ∅. Let us put two arrows labelled with B_new and F_new coming out of the current state. We then examine the Use Case and each arc in turn. Let us suppose, that JOB arrives first, i.e. B_new fires. This generates a new marking (B_dp, F_out) and set of enabled transitions {B_ab, B_go, F_new}. From the Use Case, if the JOB is at decision point but the TAG is still out of scope, then we want to decelerate the Belt, until complete rest if needed. Thus the transition B_go is undesirable: U = {B_go}. Proceeding in this way the GDS of figure 5 is built, and contains all the information about the dynamics of the two co-operating subsystems.

Co-ordination and synchronisation. For the GDS in figure 5, let us examine the set of undesirable transitions one by one. For example, B_new is not allowed to fire if and only if the marking is (B_out, F_wrap). This is achieved adding the place SP1, which is always marked except when F_wrap is marked (in fact its token is removed from the firing of F_go or F_start and F_exit replaces it). The same applies to F_new.

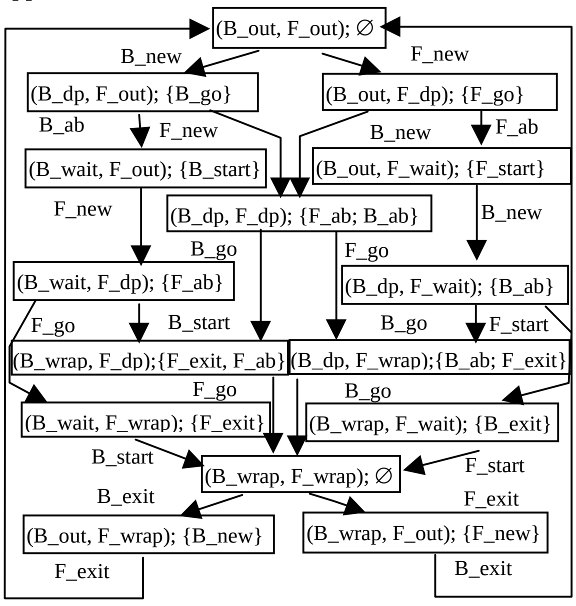

Fig. 5. GDS.

Similarly, B_ab is always an undesirable transition except when F_wrap is marked, therefore a double-sided arc between F_wrap and B_ab is added (and the same applies to F_ab). Also, B_exit is not desired before F_wrap gets marked (for the wrapping to take place the places F_wrap and B_wrap should be both marked), therefore place SP6 is added with an arc to B_exit; it is marked by the firing of F_go or F_start. Similarly SP4 is added for F_exit. Finally, B_go and B_start should fire as soon as F_dp is marked, therefore SP5 is added. SP2 is added to enable F_go and F_start as soon as B_dp is marked.

All this results in the Petri of figure 6; this graph is live and bounded and has the reachability graph shown in figure 7. The reader can notice the strong similarity with the graph of desirable case, figure 5.

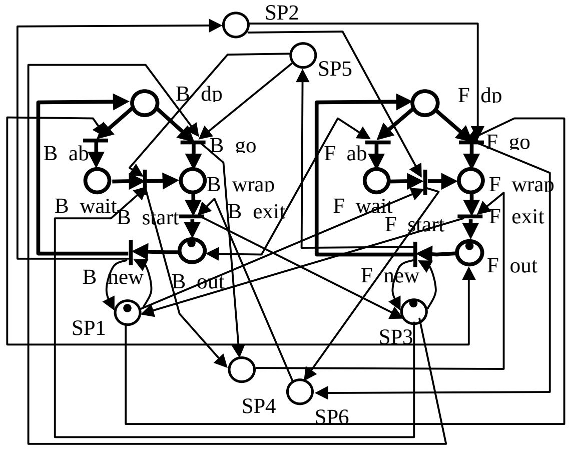

Fig. 6. Petri Net for the discrete part of the production line.

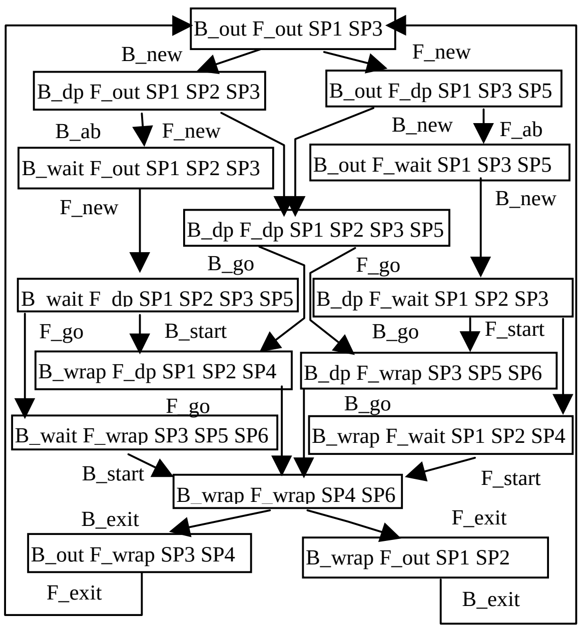

Fig. 7. Reachability graph for the petri net in figure.

4. IMPLEMENTATION

To demonstrate the design a continuous system simulation of the Film-Belt subsystem has been implemented using Matlab (vers. 5.3). To demonstrate the discrete event system involved using Stateflow: the reachability graphs of the component Petri nets were used as specifications for the design of Statechart components and interconnection of the Petri nets was modelled in using global variables which are updated when a transition takes place. The Simulink model is shown in figure 8 and the full state-chart is shown in figure 9. The belt and film systems are in two parallel sections (indicated by the dashed smoothed box). The StateFlow states of the Film and Belt during a typical synchronisation operation are shown in figure 10.

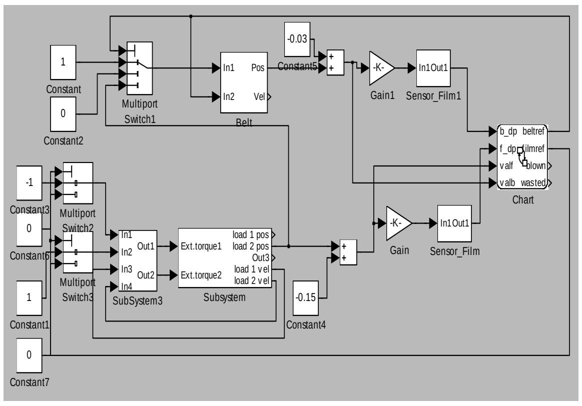

Fig. 8. Simulink scheme.

5 CONCLUSIONS

This paper has presented an integrated approach to UML for modelling and analysing discrete event controllers for real-time manufacturing systems. It has shown that Petri-net theory can be used to improve the representation and analysis of the dynamic model of such systems, making the design engineer more confident that the model accurately represents the system. It has also shown that UML use case information and compositional Petri net techniques can be used to design the coordination and synchronisation logic for large scale or compositional systems. Moreover, composite Petri-net model can be used to implement a controller based on current supervisory control theory. The technique has been illustrated by its application to a wrapping machine that forms part of a larger production line.

ACKNOWLEDGEMENTS

This work was supported by EPSRC (UK) Grant GR/L31234.

REFERENCES

Booch, G., J. Rumbaugh and I. Jacobson (1999). *The Unified Modeling Language User Guide*. Addison Wesley.

Cassandras, C.C. and S. Lafortune (1999). *Introduction to Discrete Event Systems.* Kluwer Academic Publishers.

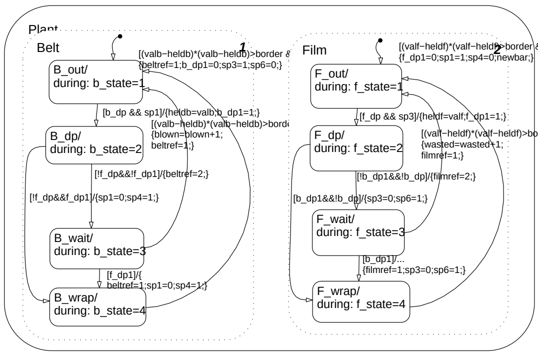

Fig. 9. Stateflow chart.

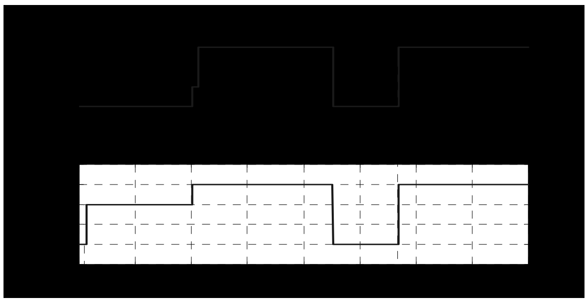

Fig. 10. States of Film and Belt in the chart: 1=_out, 2=_dp, 3=_wait, 4=_wrap.

Desrochers, A.A. and R.Y. Al-Jaar (1995). *Applications of Petri Nets in Manufacturing Systems.* IEEE Press.

Douglass, B.P. (1999). *Doing Hard Time. Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Addison Wesley.

Firesmith, D.G. (1993). *Object Oriented Requirement Analysis and Logical design: A Software Engineering Approach*. Wiley.

Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, **8**, pp. 231--274.

Juan, E.Y.T., J.J.P. Tsai and T. Murata (1998). Compositional verification of concurrent systems using Petri-net-based condensation rules. *ACM Trans. on Programming Languages and Systems*, **20(5)**, pp. 917--979.

Levenson, N.G. (1995). *Safeware, Systems safety and Computers*. Addison Wesley.

Murata, T. (1989). Petri Nets: properties, analysis and applications, *Proceedings of the IEEE*, **77(4)**, pp. 541-580.

Ramadge, P.J. and W.M. Wonham (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control & Optimization*, **25(1)**, pp. 206-230.