ENGINEER - Vol. XXXXVI, No. 02, pp. [13-33], 2013

Formal Verification Strategy for Statechart based Design of Reconfigurable Control of High Integrity Reactive Systems

S. Devapriya Dewasurendra

Abstract: Automated high integrity reactive systems required in the control of defence, automotive, rapid mass transport, manufacturing and healthcare systems, among many others, accentuate the need for formal specification and design tools. Reconfigurability by design is an added requirement given the costly development processes. Many of these systems are hybrids of discrete-event and continuous-time subsystems. Statecharts (implemented as stateflow in Matlab) are increasingly being proposed as the language of choice for the discrete-event part. However, the complex semantics of statecharts make automated formal verification difficult and hence largely an unresolved problem. Formal verification, in preference to simulation/testing, is necessary to specify these systems at the required level of integrity and to maintain traceability along the different phases of design and operation. Drawing from a number of different approaches by others to develop formal semantics at requirement and implementation levels, a first attempt was made by the author to develop a modular formal verification strategy applicable to statechart based controller specifications for complex reactive systems and an early version of the original approach, which was completely based on regular languages (hence, finite state automata) and different compositions thereof, was published in 2005.The key idea was the implementation of these composition operations through suitably interpreted port structures between pairs of automata resulting from a decomposition of the statechart. Application development based on this model has been done with collaborators on an elevator test rig built for the purpose and still continuing. While many of the underlying concepts are still valid and there is much to be learnt from their practical implementation and extension, there are fundamental limitations in this model stemming from the rather basic computational and expressive power of regular languages. In the current paper the author attempts to extend the method to context-free language based models (hence, pushdown automata, PDA) to include real-time semantics for internal (fast) events and general correctness properties expressed in temporal logic, posed as supervisory specifications of Ramage-Wonham type interpreted on the port structures between pairs of PDA whenever possible. It is shown that decidability problems limit this choice to restricted forms of context-free languages, and the development is done with Event-clock Visibly Pushdown Automata (ECVPA), which are closed under Boolean operations and determinisable. The ECVPA based verification model is presented with high-speed railway signalling as a target environment.

Keywords: Reactive Systems, Formal Verification, Statecharts, Context-free Languages, Pushdown Automata, Event-clock Visibly Pushdown Automata, Controllability, Prioritised-synchronous-composition, Real-time, Railway Signalling, Supervisory Control Theory, Control

1. Introduction

1.1.

Automated high integrity reactive systems required in the control of defence (e.g., target-tracking system), automotive (e.g., traction-control system), rapid mass transport (e.g., collision avoidance system) and manufacturing (exception handling system) need formal verification to guarantee their fault-free operation. For these complex reactive systems, which continuously react to external stimuli (called events), we need methods and tools that permit specifying them in a precise, easy and safe way, maintaining traceability along the different phases of the design that facilitate analysis and verification of behaviour. Modularity, re-configurability, *formal specification* and *compositional* formal verification are crucial considerations.

Modelling languages, like UML [36] describe

Eng. (Dr.) S. D. Dewasurendra, B.Sc. Eng. (Peradeniya), C.Eng., MIE(Sri Lanka), C.Eng., MIMechE(UK),MIEEE, M.Eng. (AIT), DEA, Ph.D.(Grenoble, Fr), Senior Lecturer, Dept. of Computer Engineering, Faculty of Engineering, University of Peradeniya, Sri Lanka.

high-level structure and behaviour, rather than implementation of solutions. Simulators can help to *validate* systems, i.e., discover design flaws, if they occur in a simulation session. Similar to testing, simulators cannot show the absence of errors. In contrast, *formal verification* establishes correctness by mathematical proof. Logically expressed properties are completely unambiguous and could thus form the basis for automated verification and validation of systems.

1.2.

The general context of formal verification in this environment has been discussed in [35] and [6] and the preferred base models are finite state structures. Traditionally the verification problem is posed as determining whether a formula φ evaluates to true or false in an interpretation κ, written $\kappa \models \varphi$, where κ represents a system modelled as a finite state structure and φ represents a correctness property of this system. This task is generally known as model checking. The model checker either confirms that φ is true in the model, which is a valuation of κ, or informs the user that it does not hold, in which case the model checker will also provide a counter-example: a run of the system that violates the property. However, in practical situations the resource requirements (execution time and memory requirement) could prohibit verifying more than an approximate model of the system. Hence, a positive outcome may not guarantee the correctness of the system any longer, and reciprocally, an error found by the model checker may be due to an inaccurate abstraction of the system. Nevertheless, this has been the preferred approach to system verification for computer scientists: for example, the model checkers SPIN [49] and NuSMV [50].

1.3.

On the other hand, the Supervisory Control Theory (SCT) for Discrete Event Dynamic Systems (DES) which integrates systems and control theory and formal computer science [2] formalises most of the applicable approaches stemming from control theoretic reasoning to this problem: Markov chains, automata, Max+ Algebra and Petri Nets. A discussion and comparison of these models can be found in [18], [19]. The basic supervisory control framework for modelling DES is an un-timed logical model that is expressed in terms of the observation and inhibition of events [2]. Within this framework, system behaviours are described by languages (i.e. sets of strings of events) and the theory seeks to determine which behaviours can be achieved through inhibition of future events by a supervising agent.

Many aspects of this model have been studied: a partial list includes observability [20], control based on partial observation [21], [22], modular control [23], decentralized control [24], non-determinism [25], timed aspects [26] and concurrency [27]. Complexity issues arise in the supervisory model due to combinatorial explosion on the number of states [28]. Hierarchical formulations for supervisory control have appeared in [29]. This body of research represents the most comprehensive set of formal verification and validation tools in controller design for asynchronous event-driven systems up to now. The base model is always the finite state machine in its different forms or equivalently, language models. However, industrial adaptations have been very limited and SCT remains mostly confined to theorists.

1.4.

The statechart model introduced by Harel [4] extends state machines along three orthogonal dimensions - hierarchy, concurrency and communication - resulting in a compact visual notation that allows engineers to structure and modularise system descriptions. In the statechart formalism event communication and data determines possible sequences of states. Often the behaviour is dependent on real-time properties and is supported by industrial simulation tools like STATEMATE [17]. The generated traces of the system model can be validated by comparing with the intuitive understanding of the system.

1.4.1. In order to formally verify a system specified with statecharts *formal semantics* are needed. The complex state hierarchies and configurations, inter-level transitions, group transitions, transition priorities and simultaneous execution of maximal non-conflicting sets of transitions present in statecharts interact in intricate and unexpected ways [5], [37] making the definition of workable formal semantics difficult and as a consequence, automated verification of models a difficult and hence, largely an unresolved problem. However, this formalism has already installed itself as one of the most powerful tools for the specification and implementation of controllers for complex reactive systems. Bhaduri and Ramesh [5] give a survey of

existing approaches to the formal verification of statecharts using model checking [6]. In [30] a formal *logical* language is used, with real-time properties expressed in *timed computation tree logic* TCTL [31]. However, these specifications form only the input to a proprietary model-checker. The basic approach followed in symbolic model checkers like SPIN and NuSPIN is to formulate logical and performance (temporal) specifications as well-formed-formulas (wff) and to test them using Binary Decision Diagram (BDD) on system models developed as Kripke structures (a labelled transition system).The common problems here are the limited semantics captured in the translation of statecharts to Kripke structures and state space explosion in the resulting flat model.

1.4.2. In [17] the author proposed a modular formal verification strategy applicable to statechart based controller specifications for complex reactive systems which was completely based on regular languages(hence, finite state automata) and different compositions thereof. The key idea was the implementation of these composition operations through suitably interpreted port structures between pairs of automata resulting from a decomposition of the statechart. Application development based on this model has been done with collaborators on an elevator test rig built for the purpose and still continuing [58, 59, 60]. While most of the underlying concepts are still valid and there is much to be learnt from their practical implementation and extension, there are fundamental limitations in this model stemming from the rather basic computational and expressive power of regular languages, being the lowest in Chomsky hierarchy. In the current paper the author attempts to extend the method to context-free language based models (hence, pushdown automata, PDA) to include real-time semantics for internal (fast) event handling and general correctness properties expressed in temporal logics, posed as supervisory specifications of Ramage-Wonham type whenever possible. Then the verification task reduces to one of checking the controllability of these supervisory specifications for PDA interpreted as local plant models. As an outcome of this effort it is shown that decidability problems limit this choice to restricted forms of context-free languages, and that Event-clock Visibly Pushdown Automata (ECVPA), which are closed under Boolean operations and determinisable, are arguably the best available option. Hence, the development was done with ECVPA. The resulting compositional verification method is presented with high-speed railway signalling as a target application environment.

The remaining chapters are organised as follows: Ch. 2 gives a very brief introduction to Supervisory control theory for DES, followed by Establishing the need for a formal verification methodology applicable to statecharts in Ch. 3. Outstanding issues and available options for developing a suitable formal verification methodology are given in Ch. 4.Ch. 5 presents the proposed modular verification methodology and its application to railway signalling. Ch. 6 concludes the paper.

2. Supervisory control Theory for Discrete Event Systems (DES)

A plant to be controlled is modelled as a language generator defined as a 5-tuple [2],$G = (Q, \Sigma, \delta, q_o, Q_m)$Where,$\Sigma$ is the alphabet of events; Q is the set of states;q_o is the initial state;Q_m is the set of final or marker states; and$\delta$is a *partial function* (pfn)

$\delta: Q \times \Sigma \rightarrow Q$ (pfn) (2.1)

The partial function, δ, is extended to event strings as $\delta(q, \varepsilon) = q$; $\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)$, provided $q' := \delta(q, s)!$ and $\delta(q', \sigma)!$.

$L(G)$ is the *language generated* by G. $L_m(G)$ is the *language marked* by G. Some of the events in the alphabet Σ can be controlled (hence, called controllable), denoted Σ_c, while the other events are uncontrolled (or uncontrollable), denoted Σ_u: $\Sigma = \Sigma_c \cup \Sigma_u$. The *reachable subset* of G is $Q_r = \{q \in Q \mid (\exists s \in \Sigma^*)\delta(q_o, s) = q\}$. G is *reachable* if $Q_r = Q$. The *coreachable subset* is $Q_{cr} = \{q \in Q \mid (\exists s \in \Sigma^*)\delta(q_o, s) \in Q_m\}$. Gis *coreachable* if $Q_{cr} = Q$. G is said to be *nonblocking* if $L(G) = \overline{L}_m(G)$, which means that any string that can be generated by G is a prefix of a marked string of G. G is *trim* if it is both reachable and coreachable. A *supervisory control* for G amounts to enabling some controllable events at each state: i.e., a map $V: L(G) \rightarrow \Gamma$. Each subset of enabled events (in a given state) is a *control pattern*; and the set of all control patterns is $\Gamma = \{\gamma \in Pwr(\Sigma) \mid \gamma \supseteq \Sigma_u\}$. The pair (G,V) will be written as V/G to suggest 'G under the supervision of V': The controller tracks the state changes in the system and at each new state calculates a set of enabled events (and a set of disabled events).

The *closed behaviour* of V/G is defined to be the language $L(V/G)$ (2.2)

The *marked behaviour* of V/G is $L_m(V/G) = L(V/G) \cap L_m(G)$. V is *nonblocking* over G if

$\overline{L}_m(V/G) = L(V/G)$. (2.3)

A language $K \subseteq \Sigma^*$ is *controllable* with respect to G if $(\forall s, \sigma)\ s \in \overline{K}, \sigma \in \Sigma_u$ and $s\sigma \in L(G) \Rightarrow s\sigma \in \overline{K}$, which says that K is controllable if no string in $L(G)$ that is already a prefix of K when followed by an uncontrollable event in G, leaves the prefix set of K: the prefix closure $\overline{K}$ is invariant under the occurrence in G of uncontrollable events. More concisely, K is controllable if and only if

$\overline{K}\,\Sigma_u \cap L(G) \subseteq \overline{K}$ (2.4)

Let $K \subseteq L_m(G)$, $K \neq \phi$. There exists a nonblocking supervisory control V for G such that $L_m(V/G) = K$ if and only if K is controllable with respect to G, and K is $L_m(G)$-*closed*. (2.5)

Let $E \subseteq \Sigma^*$. Then the set of all sublanguages of E that are controllable with respect to G is given by $C(E) = \{K \subseteq E \mid K \text{ is controllable with respect to } G\}$. $C(E)$ always contains a unique supremal element, which we denote as *sup* $C(E)$.

Let $E \subseteq \Sigma^*$ be $L_m(G)$-marked, and let $K = sup\ C(E \cap L_m(G))$. If $K \neq \phi$, there exists a non-blocking supervisory control (NSC) V for G such that $L_m(V/G) = K$.

In modular synthesis, even if separate local supervisor actions were non-blocking, concurrent action of these supervisors may lead to blocking or conflict. Therefore, it is important to check for conflict between supervisors. If supervisors S_i, S_{i+1}, S_{i+2}, S_{i+3}, ... are non-blocking supervisors for generator G then the concurrent action $S_i \wedge S_{i+1} \wedge S_{i+2} \wedge S_{i+3} \wedge \ldots$ is non-blocking if and only if the languages $L_c(S_i/G), L_c(S_{i+1}/G), \ldots$ are non-conflicting, in other words [33]:

$$\overline{L_c(S_i/G) \cap L_c(S_{i+1}/G) \cap L_c(S_{i+2}/G) \ldots} = \overline{L_c(S_i/G)} \cap \overline{L_c(S_{i+1}/G)} \cap \overline{L_c(S_{i+2}/G) \ldots} \quad (2.6)$$

A more useful result related to modular control, in the author's opinion, is given by the combination of the following results:

(2.5) above and propositions 5.1 and 5.2 of [2] taken with the fact that any two closed languages are non-conflicting. (2.7)

The importance of (2.7) derives from the fact that,

if K_1 and K_2 are two closed and controllable languages for a plant G, then the existence of non-blocking modular supervisors S_1 and S_2, implementing K_1 and K_2 is guaranteed and that the controllability of $K_1 \cap K_2$ for G and the existence of a non-blocking supervisor S_{12} to implement $K_1 \cap K_2$ are also guaranteed. (2.8)

In general, local decisions of local supervisors are fused by a decision fusion rule in order to arrive at global control action of a decentralised supervisor. The permissive control of SCT implies a conjunctive fusion rule where a controlled event is enabled by default, and is disabled if at least one local supervisor decides to disable it. Taken as general rule this appears to pose a difficult problem for event forcing: event forcing is necessary in implementing logic control, particularly in real-time and safety-critical contexts. An event perfectly legally forced by one module could result in an illegal string for some other module. However, there are many controllable events that can safely be disabled by default and enabled if at least one supervisor decides to enable it [34]: disjunctive fusion rule. The condition (2.6) applies only if all controllable events are subject to the conjunctive decision fusion rule. Co-observability along with controllability have been proposed in [40] as a condition necessary and sufficient for guaranteeing (2.6). The D&A co-observability of [34] is a direct extension of this to include mixed fusion rules (disjunctive and conjunctive).

D&A co-observability, controllability and $L_m(G)$-*closure* are necessary and sufficient conditions for the existence of non-blocking decentralised supervisors in this latter case. (2.9)

A polynomial time algorithm for finding whether there is a partition of controllable events into conjunctive and disjunctive sets in such a way that co-observability is satisfied and if such a partition exists, a polynomial time algorithm to compute it too are given in [34]. (2.10)

Remark II.1: A common flaw we see in the approaches of both [40] and [34] is the *feasibility* assumption: i.e., that the state does not change when an unobservable event happens.

The case where $G = \|_{i=1}^{n} G_i$ with local supervisors acting on local plant models G_i to result in a controlled behaviour of the form $\bigcap_{i=1}^{n} L_c(S_i/G_i)$ has only been studied under very strong structural assumptions on the separability of specifications and disjoint local plant alphabets [38], [39]. The reader can get a good understanding of the basics of Supervisory Control Theory for DES from [1] and [2].

Hence, from the point of view of developing compositional procedures for checking controllability of logical and behavioural specifications on a distributed plant model, which forms the central idea in the proposed formal verification model, the current technology in supervisory control theory does not provide direct answers, particularly in view of the problems related to formal verification of statecharts as identified in the section 1.4.1 above and further demonstrated in Ch. 3 below. However, as will be seen in the suite of this paper, we manage to develop an effective strategy to address these issues based on a combination of these and some other results related to the extension of SCT to non-regular languages.

Now we will take a closer look at individual features of statecharts and the verification challenges they pose.

3. Establishing the need for a formal verification methodology applicable to statecharts

3.1. The complex features of statecharts interact in intricate and unpredictable ways. Hence it is extremely difficult to provide a coherent formal semantics to these semantically rich features. This has resulted in a large number of proposals for statechart variants and their formal semantics [5]. In the model checking framework, which is by far the verification and validation method with widest acceptance in industry as well as research community, the system is represented as a transition system (equivalently, a Kripke structure [32]). A transition system is defined as a tuple (Q, ι, R), where Q is the set of *states*, usually specified by assignments of values to a set of variables V; ι is a set of initial states (expressed as predicates on V); and R is the transition relation, usually expressed by predicates containing unprimed and primed variables from V for the pre- and post-state. Kripke structures have an additional component: a labelling of each state with the set of propositions that hold in that state; in a transition system, the set of propositions that hold in a state is given by the set of predicates on state variables that evaluate to true in that state. Binary Decision Diagrams (BDD) have been extensively used in modern symbolic model checkers like NuSMV to ascertain whether logical and temporal (behavioural) specifications can be guaranteed to hold. While BDD representations render many verification problems in this context decidable, as described in the rest of the chapter, the key problems of, difficulty in faithfully capturing the complex informal semantics of statecharts in Kripke structures and state explosion of Kripke structures remain.

Some features of statecharts in its *unrestricted* form that complicate this interpretation of statecharts as transition systems, making automated verification of statecharts models a difficult problem are, State Hierarchy, Non-compositionality resulting from inter-level transitions, Conflicting Transitions and Transition Priority, Concurrency, Communication, History and Models of Time.

3.2. Attempts have been made from a practical implementational perspective to formulate *formal* semantics that would restrain or remove some of these features as required, without compromising the representational/computational power necessary for unambiguously specifying the requirements of a target application area: real-time systems, for instance. In this approach, one starts from what is required and attempts to build the formal semantics that would guarantee unambiguous expression of the design specifications and the realizability of the latter, and do this by borrowing the semantical possibilities offered by statecharts. This would permit one to retain the essential subset of features (targeted for the critical application area) that could be captured in a rigorous formal semantics. Notable in this kind of approach is the work reported in [51] related to real-time statechart semantics. The formal semantics would permit one to form well formed formulas, but it is also important to know decidability properties in the formalism when the focus is formal verification. In this regard, the formal semantics proposed for restricted statecharts derive basically from timed automata of Alur-Dill [52] type, for which most decision problems like language inclusion are undecidable. On the other hand, there are restricted forms of time constraints for which the reachability problem remain decidable. Likewise, compositionality results are available for restricted forms of timed input/output automata. We shall now look at key features of statecharts from the perspective of semantics necessary for formal verification of high integrity reactive systems specified with them.

3.3. State Hierarchy: The straightforward way to represent a statechart as a transition system is to flatten its hierarchy. While this would permit translation of the statechart to the input languages of standard model checking tools, the resulting exponential blow-up of states can limit the size of models that can be tested. Hence, there is a need for facilitating modular/compositional testing/verification.

Non-compositionality: inter-level transitions A syntax-directed, hierarchical semantics and translation scheme for statecharts is crucial for efficient verification of their correctness. A *compositional semantics* would interpret the meaning of a composite statechart in terms of the semantics of its constituent components, without having to consult their internal structure. The semantics of inter-level transitions violate the state encapsulation hierarchy and hence compositionality. The most comprehensive treatment of compositional semantics for statecharts has been by Damm et al [61], which focuses on asynchronous semantics or super-step semantics. A super-step consists of a chain of reactions (called steps) of the system being controlled to an external stimulus. Their compositional semantics developed in terms of compositional synchronous transition systems distinguish between external and internal events and provide them with different semantics. They consider that internal variables of two systems as disjoint: is this realistic always? What if an internal variable of one system is an external variable for the other? Notably, too they assume that each object has complete knowledge about the variables generate by other systems that can have any effect on it and that all local computations are carried out with this knowledge. This is almost like centralised control from the point of view of any given system! However, with these and other assumptions it becomes possible to perform compositional verification on their well-formed statecharts using symbolic model checking. Huizing et al [53] also have proposed a compositional semantics for statecharts using their Unvs or incomplete statecharts as constructs and the notion of observable behaviour they introduce. We have not gone deep into this, but there seem to be interesting possibilities stemming from this important work: we intend to pursue this line in our future work.

Conflicting Transitions and Transition Priority: Two transitions are in conflict if there is a common state shared by their source states. If two conflicting transitions are at the same level of the state hierarchy the result is *non-determinism*, which is an undesirable phenomenon in the execution of a reactive system. Automatic detection of deadlock is also a challenge, since the enabledness of transitions depends on conditions on data values, and the problem is undecidable in general.

Concurrency: A faithful modelling of concurrency in statecharts (either at the level of concurrent sub-states or multiple statecharts) requires the scheduling of concurrent execution of transitions at a fine level of granularity. Issues like multithreaded and single-threaded execution complicate the matter. Most modelling formalisms have two principal means of handling concurrency: by synchronous execution and interleaving, with very little control in mixing the two.

Communication: Several dimensions of communication mechanisms can be identified in statechart variants: broadcast versus point-to-point, synchronous versus synchronous, instantaneous versus timed. Modelling of asynchronous communication in transition systems causes subtle problems, because the levels of granularity of interleaving in statecharts may be different from the on eemployed by the modelling language, which in most cases is fixed. Asynchronous models also lead to a blow-up in the size of the state space, making model checking impractical for real life examples. Eshuis et al [62] claim that the following choices have to be made for any semantics of statecharts, regardless of semantic level or design paradigm:

A clock-synchronous and/or clock-asynchronous semantics. In the clock-synchronous semantics, the system starts processing its input only at the tick of the clock. In the clock-asynchronoussemantics, the system starts processing its input as soon as it receives it.

Synchronous or asynchronous communication. In synchronous communication, the caller must wait until the callee has finished processing the communication. In asynchronous communication, the caller continues without waiting for the receiver to finish processing the communication.

They show in their requirements-level semantics that, synchronous communication is only possible with a clock-asynchronoussemantics. Their semantics are defined on a timed variant of Labelled Transition Systems called Clocked Labelled Kripke Structures (CLKS).

History: The statecharts model has two history connectors: *H* (shallow history) and *H**(deep history). When entering an OR state by shallow history, the sub-stateentered is the one most recently visited, and on entry by deep history, the basic configuration (including all recursively contained sub-states) last visited relative to the OR state, is entered. The modelling of deep history implies all state configurations are stored in a variable that retains its value between transitions, whereas for shallow history only the state variable corresponding to the top level sub-state last visited should be retained. This complicates the treatment of variables in the model, with some variables requiring special treatment.

Models of Time: The definition of what constitutes a step is central to the semantics of statecharts. The synchronous semantics of statecharts referred to above is simpler to model using transition systems. The greediness of transition execution inherent in the asynchronous semantics makes it difficult to model. As explained above, Eshuis et al [62]present two different ways of executing a step. In the clock-synchronoussemantics, a step is executed when the clock ticks. In the clock-asynchronous semantics, a step is executed immediately when new events arrive. Then the system becomes unstable and reacts infinitelyfast to become stable again. This corresponds to the execution of a super-step in [61] and the synchronous assumption in synchronous languages. Since these are requirement level semantics for real-time systems, they are of relevance to our work. However, the CLKS on which they have developed their semantics imply all the limitations attributed to Kripke structures above.

Thus the infeasibility of formally verifying monolithic control specifications using the complete semantic richness of statecharts and the need for modular specification and verification methodologies applicable to statechart based controllers for reconfigurable reactive systems is established. The almost total dependence on model checking is seen as a consequence of early realisation that the satisfiability of behavioural properties stemming from rich statechart semantics is most often undecidable:this makes it unrealistic to expect formal proofs for large-scale monolithic specifications.

4. Outstanding issues and available options for developing a suitable formal verification methodology

The rest of the section is devoted to the progressive demonstration of the outstanding issues in the development of aformal verification methodology through critical examination of the available options. The solution proposed in the next section will be a natural development from the analysis here.

Remark IV.1: The main concern is how to find a way to bring in compositionality and then to develop a modular verification strategy in the presence of inter-level transitions.

Drusinsky and Harel [7] develop the statechart into a tree of interconnected finite state structures (FSMs) under the implicit assumption that there are no real outputs on states or transitions, in which case all input events are externally generated. However, their objective in this decomposition was hardware optimisation (substrate layout design to obtain reduced area and power consumption) for a PLA implementation of controllers specified using statecharts. They use the fact that the area of PLA is mainly determined by the number of minterm lines which are in the order of n^2, where n is the number of states, with one minterm for each transition of an FSM [7]. They show that using their decomposition the number of minterms would be determined by the largest submachine (sub-FSM) and intra-machine connections. In this manner, by keeping the sub-machine size small they achieve great layout economy. In this design process their design choices do not seem to be affected by real outputs (events) on states or transitions which could be inputs (internal events, also called fast events in [61]) to trigger conditions of other transitions. In contrast, these internal events are a major concern in our design of a compositional formal verification scheme (the earlier regular language based proposal by the author did not consider these internal events).

Remark IV.2: The approach would look more appealing if internally generated event handling capacity could be incorporated into this development.

Remark IV.3: This seems to be possible using the results of Tiwari [41] in mapping Simulink

stateflow models (Matlab implementation of statecharts) to a set of automata communicating through a global stack for events. The choice of a global stack is justified by the stateflow semantics which treat events sequentially, one at a time (in contrast to richly parallel semantics implied by the original statechart model). The preoccupation in [41] has been the provision of formal semantics to stateflow which lacked any. The unlimited memory provided by the stack is all what they needed to store the internally generated events and account for them in a pushdown-like interpretation. This, however, does not elevate their model to a communicating pushdown system (even though they call it so), because a pushdown automaton, by definition, should have its own infinite stack, not one shared with others. Nevertheless, for the restricted semantics implemented by stateflow, this scheme provides higher expressive power than a regular language based interpretation. However, in order to capture parallel semantics of statecharts it would be necessary to manage the internally generated events separately for AND states: i.e., using separate stacks. This suggests a set of communicating pushdown automata (PDA) with a single stack for each OR state and individual stacks for component FSMs of AND states. This would then make it possible to use the higher computing power of context-free grammars instead of regular grammars, and hence, first-order logic instead of propositional logic. While this is a plus point, how to capture synchronous interpretation of certain transitions and asynchronous firing of others on some of these PDA is not very clear. Also, the results known to us on the application of formal verification tools to a plant modelled as a PDA are limited to reachability analysis [41], [46].In what concerns model checking approaches, algorithms for various temporal logics have been proposed for pushdown systems: while the model-checking problem for branching-time logics is computationally intractable, linear-time logic can be solved in polynomial time (for a fixed formula). Since even these results, limited as they are, are related to monolithic PDA, implementation of statecharts as communicating PDA does not seem to be very promising from a compositionality point of view.

Remark IV.4: Griffin [46] considers a regular plant language, L and a context-free specification language, S, and manages to show that the controllability of $L \cap S$ is decidable. Now, $L \cap S$ is a context-free language for L context-free and S regular [54]. This is an important result. Then there is the work by Alur et al on *visibly pushdownautomata*(VPA) [55] and *event-clock automata* [56] and *event-clock visibly pushdown automata*(ECVPA) by Tang et al [57] which provide compositional semantics for a restricted, but adequately powerful version of pushdown automata in the context of the control of reactive systems.

A (non-deterministic) visibly pushdown automatonon finite words is defined over a partitioned input alphabet $\Sigma_c \cup \Sigma_r \cup \Sigma_l$, where Σ_c is a finite set of calls, $\Sigma_r$is a finite set of returns and $\Sigma_l$is a finite set of local actions. The restriction is such that it pushes onto the stack only when it reads a call, it pops the stack only at returns, and does not use the stack when it reads local actions. The input hence controls the kind of operations permissible on the stack. However, there is no restriction on the symbols that can be pushed or popped [55]. This mode of operation is adequate for the present task environment.

VPA are closed under Boolean operations, which property is fundamental in terms of compositionality of models. Further, the inclusion problem (i.e., $L(A) \subseteq L(B)$) is decidable for them: this in fact is a required property for formal verification of the models, because the verification problem amounts to determining whether the plant language is included in a given specification language.

Event-clock automata of Alur et al isa determinizable class of timed automata [56] constructed by restricting the use of clocks in timed automata. The clocks of an event-clock automaton have a fixed, predefined association with the symbols of the input alphabet. The event-recording clock, x_a of the input symbol a is a history variable the value of which always equals the time of the last occurrence of a relative to the current time; the event-predicting clock, y_a of a is a prophecy variable the value of which always equals the time of the next occurrence of a relative to the current time (if no such occurrence exists, then the clock value is undefined). The class of event-clock automata is sufficiently expressive to model real-time systems with finite control, and to specify common real-time requirements. Then, $C_\Sigma = \{x_a \mid a \in \Sigma\} \cup \{y_a \mid a \in \Sigma\}$is the set of event-recording and event-predicting clocks. For each position j of a timed word $\bar{w}$, the clock-valuation function$\gamma_j^{\bar{w}}$, then, is a mappingfrom $C_\Sigma$to $R_{\geq 0}^{\Sigma}$. The clock constraints, $\Phi(C_\Sigma)$, compare

clock values to rational constants or to the special value $\perp$ which indicates the absence of a future occurrence of a. It can also be a Boolean combination of such comparisons.For instance, the hard real-time requirements that "every request is followed by a response within 3 seconds" and that "every two consecutive requests are separated by at least 5 seconds" can be expressed using event-clock automata.

An event-clock visibly pushdown automaton (ECVPA) [57] on finite timed words over $\Sigma$is a tuple $M = (Q, \Sigma, Q_0, \Gamma, \Delta, F)$, where Q is a finite set of locations, $Q_0 \subseteq Q$ is a finite set of initial locations, $\Gamma$is a finite stack alphabet that contains a special symbol $\perp$(bottom-of-stack symbol), $F \subseteq Q$ is a set of final locations, and $\Delta = \Delta_c \cup \Delta_r \cup \Delta_l$ is the transition relation:

$\Delta_c \subseteq Q \times \Sigma_c \times \Phi(C_\Sigma) \times Q \times (\Gamma \setminus \{\perp\})$ is a push-transition relation

$\Delta_r \subseteq Q \times \Sigma_r \times \Phi(C_\Sigma) \times \Gamma \times Q$ is a pop-transition relation

$\Delta_r \subseteq Q \times \Sigma_l \times \Phi(C_\Sigma) \times Q$ is an internal-transition relation.

The intuitive meaning of the transition relation is as follows:

$(q, a, \varphi, q', \gamma) \in \Delta_c$ is a push-transition, where on reading a when the clock valuation satisfies the clock constraint, φ, the symbol γ is pushed onto the stack and the location changes to q'.

$(q, a, \varphi, \gamma, q') \in \Delta_r$ is a pop-transition, where on reading a when the clock valuation satisfies φ, γ is popped from the stack, the location q changes to q'(if $\gamma = \perp$, it is read but not popped).

$(q, a, \varphi, q') \in \Delta_l$ is an internal-transition, where the location, on reading a when the clock valuation satisfies φ, changes from q to q' without stack operations.

The class of ECVPA is expressive enough to specify common context-free real-time properties such as "if p holds when a procedure is invoked, then the procedure must return within d time units and q must hold at the return state". Besides, the class of ECVPAs is closed under all Boolean operations.

Remark IV.5: Again it should be mentioned that Simulink stateflow has restricted semantics and Tiwari's treatment which processes only one input event at a time has been demonstrated to be adequate for the purpose of doing reachability analysis on these models. Hence, our Remark IV.3 above could be used to develop more realistic simulation environments: with more than one event stack, for instance. This will be a future axis of investigation to be undertaken in the present research programme.

Remark IV.6: On the other hand, in view of the Remark IV.3 above, another option would be to handle internal events in infinite queues as in Kahn process networks. This seems to be appropriate for addressed communication, but in order to handle event broadcast, it would be necessary to design networks of queues with multiple servers: in fact, theoretically the number of servers for each queue would be the total number of nodes (individual FSMs) in the network. This will be an approach closer to the synchronous interpretation of statecharts and hence, the synchronous semantics as seen in synchronous languages like Esterel [47]. This would be different from events generated and queued in[63].This is the second axis intended to be pursued as future work.

The ground work established through the above remarks can now be used to discuss issues related to the particular architectural choices to be made in this research. Certain elementary developments based on these choices will also be presented alongside when it is convenient to do so.

In the methodology of [7] each state, at each non-atomic level of the statechart hierarchy is represented by a machine implementing the FSM corresponding to its substates on the next immediate level. In the current adaptation the FSMs are transformed into ECVPAs. To illustrate this process we use the statechart in Fig. 6. Fig. 10 gives the ECVPA Speed_Ctrlsubstate. Part of the resulting machine interconnection scheme is given in Fig. 13.

Remark IV.7: The salient aspects of the original decomposition in [7] and used in [17] concern state and event assignment and the introduction of an ***Idle*** state for each FSM. An event ***entered*** is created by the machine one level higher in the hierarchy when it reaches state *X*. The ***left*** signals are the duals of the ***entered*** signals, and notify the lower level machines to move into their ***Idle*** state. The ***leave*** signals are created by the lower level states, to notify their predecessors about their termination. Similarly, the ***enter X*** signal is created by a high-level state when one of its sub-states (that is not the default) is required to start operating in the *X*. In [17] these were termed

as virtual events, $\sum_{vir}^{i}$ for a given module *i*. An immediate consequence of this identified in [17] was that in any modular approach for verification of this implementation, the modules will now have *unequal event alphabets*.

The possibility of casting this decomposition of statecharts (with restricted semantics, though) as communicating FSM's was tempting. In this respect the work of Endsley et al [13] is relevant in that they interpret communication channels between pairs of FSM as port structures modelled using notions of controllability of events from SCT; but this work remains at an empirical level with very little support from the formal tools of SCT. Their modular synthesis results rest on the restrictive assumption that there can be no shared triggers (events) or shared responses (actions) between modules. They manage to bypass all the complex issues related to event forcing by pre-empting (through this assumption) the enabling of the same event by more than one module.

Remark IV.8: Also, their approach does not pay any attention to non-blocking behaviour of the composed system, and hence is not compositional. Nevertheless, if presented with sufficient rigour, the potential of such an interpretation of communication channels became clearly visible.

Remark IV.9: Immediately then the issue of handling controllability of internal and virtual events had to be addressed: the same event being an output for one FSM and an input to another. In order to stay within SCT, it became necessary to model the problem in terms of local plant models, G_i, with different controllability sets of events. As shown in section 2 above, this case has only been studied under very restrictive assumptions of separability of supervisory specification and/or disjoint local plant alphabets [38, 39]. Existence of non-blocking solutions for the general case is undecidable.

Now we shall examine problems arising from non-deterministic behaviour of the proposed system models. A new model has been proposed in [13] based on the prioritized synchronization and trajectory models introduced by Heymann [14]. In this new type of interconnection, called prioritized synchronous composition (PSC), each system component is assigned a priority set of events. The following definition introduces the notion of prioritised synchronisation between two Non-deterministic State Machines (NSM).

Definition 1. Let $P = (X_P, \Sigma, \delta_P, x_P^0)$ and $Q = (X_Q, \Sigma, \delta_Q, x_Q^0)$ be two NSMs. Let A, B$\subseteq \Sigma$ be the priority sets of P and Q, respectively. Then the Prioritised Synchronous Composition (PSC) of P and Q, denoted $P \|_{A\,B} Q$, is the NSM defined as $P \|_{A\,B} Q := (X, \Sigma, \delta, x^0)$, where $X := X_P \times X_Q$, $x^0 := (x_P^0, x_Q^0)$, and the transition function $\delta : X \times \overline{\Sigma} \to 2^X$ is defined as:

$\forall x = (x_q, x_q) \in X, \sigma \in \Sigma$:

$$\delta(x,\sigma) := \begin{cases} \delta_P(x_p,\sigma) \times \delta_Q(x_q,\sigma) & \text{if cond.1} \\ \delta_P(x_p,\sigma) \times \{x_q\} & \text{if cond.2} \\ \{x_p\} \times \delta_Q(x_q,\sigma) & \text{if cond.3} \\ \phi, otherwise \end{cases}$$

where, cond.1: $\delta_P(x_p,\sigma) \neq \phi, \delta_Q(x_q,\sigma) \neq \phi$;

cond.2: $\delta_P(x_p,\sigma) \neq \phi, \delta_Q(x_q,\sigma) = \phi, \sigma \notin B$;

cond.3: $\delta_Q(x_p,\sigma) \neq \phi, \delta_P(x_P,\sigma) = \phi, \sigma \notin A$.

$\delta(x,\varepsilon) := [\delta_P(x_p,\varepsilon) \times \{x_q\}] \cup [\{x_p\} \times \delta_Q(x_q,\varepsilon)]$.

If P represents an uncontrolled plant, Q a supervisor, and $P \|_{A\,B} Q$ the controlled plant, then the set of *controlled* events is $A \cap B$: the set of *uncontrolled* events is $A - B$; the set of *driven* events is $B - A$; and the set $\Sigma - (A \cup B)$ is assumed to be empty.

The following theorem solves the supervisory control problem in the context of PSC.

Theorem 1. [11] Let P be an NSM with event set Σ, $(A \cup B) = \Sigma$, and let K be a nonempty prefix-closed sublanguage of $L(P^{B-A})$, Then there exists a deterministic state machine S such that $L(P_A \|_B S) = K$ if and only if K is $(L(P^{B-A}), A - B)$ - controllable, in which case S can be chosen as the deterministic state machine such that $L(S) = K$, where, for $\Sigma_d \subseteq \Sigma$, $\det(\Sigma_d^*)$ denotes the deterministic state machine with one state and self-loops labelled by every event in Σ_d. The *augmentation of P by* Σ_d, denoted P^{Σ_d}, is defined to be the NSM $P^{\Sigma_d} := P_\phi \|_\phi \det(\Sigma_d^*)$.

as **virtual events**, $\sum_{vir}^{i}$ for a given module *i*. An immediate consequence of this identified in [17] was that in any modular approach for verification of this implementation, the modules will now have *unequal event alphabets.*

The possibility of casting this decomposition of statecharts (with restricted semantics, though) as communicating FSM's was tempting. In this respect the work of Endsley et al [13] is relevant in that they interpret communication channels between pairs of FSM as port structures modelled using notions of controllability of events from SCT; but this work remains at an empirical level with very little support from the formal tools of SCT. Their modular synthesis results rest on the restrictive assumption that there can be no shared triggers (events) or shared responses (actions) between modules. They manage to bypass all the complex issues related to event forcing by pre-empting (through this assumption) the enabling of the same event by more than one module.

Remark IV.8: Also, their approach does not pay any attention to non-blocking behaviour of the composed system, and hence is not compositional. Nevertheless, if presented with sufficient rigour, the potential of such an interpretation of communication channels became clearly visible.

Remark IV.9: Immediately then the issue of handling controllability of internal and virtual events had to be addressed: the same event being an output for one FSM and an input to another. In order to stay within SCT, it became necessary to model the problem in terms of local plant models, G_i, with different controllability sets of events. As shown in section 2 above, this case has only been studied under very restrictive assumptions of separability of supervisory specification and/or disjoint local plant alphabets [38, 39]. Existence of non-blocking solutions for the general case is undecidable.

Now we shall examine problems arising from non-deterministic behaviour of the proposed system models. A new model has been proposed in [13] based on the prioritized synchronization and trajectory models introduced by Heymann [14]. In this new type of interconnection, called prioritized synchronous composition (PSC), each system component is assigned a priority set of events. The following definition introduces the notion of prioritised synchronisation between two Non-deterministic State Machines (NSM).

Definition 1. Let $P = (X_P, \Sigma, \delta_P, x_P^0)$ and $Q = (X_Q, \Sigma, \delta_Q, x_Q^0)$ be two NSMs. Let A, B $\subseteq \Sigma$ be the priority sets of P and Q, respectively. Then the Prioritised Synchronous Composition (PSC) of P and Q, denoted $P \|_{A\,B} Q$, is the NSM defined as $P \|_{A\,B} Q := (X, \Sigma, \delta, x^0)$, where $X := X_P \times X_Q$, $x^0 := (x_P^0, x_Q^0)$, and the transition function $\delta : X \times \overline{\Sigma} \to 2^X$ is defined as:

$\forall x = (x_q, x_q) \in X, \sigma \in \Sigma$:

$$\delta(x,\sigma) := \begin{cases} \delta_P(x_p,\sigma) \times \delta_Q(x_q,\sigma) & \text{if cond.1} \\ \delta_P(x_p,\sigma) \times \{x_q\} & \text{if cond.2} \\ \{x_p\} \times \delta_Q(x_q,\sigma) & \text{if cond.3} \\ \phi, otherwise \end{cases}$$

where, cond.1: $\delta_P(x_p,\sigma) \neq \phi, \delta_Q(x_q,\sigma) \neq \phi$;

cond.2: $\delta_P(x_p,\sigma) \neq \phi, \delta_Q(x_q,\sigma) = \phi, \sigma \notin B$;

cond.3: $\delta_Q(x_p,\sigma) \neq \phi, \delta_P(x_P,\sigma) = \phi, \sigma \notin A$.

$\delta(x,\varepsilon) := [\delta_P(x_p,\varepsilon) \times \{x_q\}] \cup [\{x_p\} \times \delta_Q(x_q,\varepsilon)]$.

If P represents an uncontrolled plant, Q a supervisor, and $P \|_{A\,B} Q$ the controlled plant, then the set of *controlled* events is $A \cap B$: the set of *uncontrolled* events is $A - B$; the set of *driven* events is $B - A$; and the set $\Sigma - (A \cup B)$ is assumed to be empty.

The following theorem solves the supervisory control problem in the context of PSC.

Theorem 1. [11] Let P be an NSM with event set Σ, $(A \cup B) = \Sigma$, and let K be a nonempty prefix-closed sublanguage of $L(P^{B-A})$, Then there exists a deterministic state machine S such that $L(P_A \|_B S) = K$ if and only if K is $(L(P^{B-A}), A - B)$ - controllable, in which case S can be chosen as the deterministic state machine such that $L(S) = K$, where, for $\Sigma_d \subseteq \Sigma$, $\det(\Sigma_d^*)$ denotes the deterministic state machine with one state and self-loops labelled by every event in Σ_d. The *augmentation of P by* Σ_d, denoted P^{Σ_d}, is defined to be the NSM $P^{\Sigma_d} := P_\phi \|_\phi \det(\Sigma_d^*)$.

In view of the Remarks numbered II.1 to IV.9 above, a comprehensive formal verification process was developed for statechart models of complex reactive systems, which is modular and incremental. The approach taken is to ensure safe behaviour for all possible temporal orderings of uncontrollable events at any state in the plant dynamics by enabling/disabling controllable events according to a joint conjunctive/disjunctive fusion strategy. An early version of this process based on regular languages (and hence, finite state automata), where internal events and real-time issues arising from them were not taken into consideration, was first presented in [16] and [17]. The following gives a considerably improved version of the methodology, where these have been incorporated to provide a compositional formal verification tool which goes beyond the current approaches cited in the previous chapters.

5. Proposed modular verification methodology and application to railway signalling

(i) The proposed methodology starts with the scheme of Drusinsky and Harel [7] to develop the statechart into a tree of interconnected finite state machines (FSM). Each state, at each non-atomic level of the statechart hierarchy, is represented by a machine implementing the FSM corresponding to its sub-states on the next immediate level.

(ii) Then these automata are augmented with stacks to handle internal events (we include all fast-events of [61] and [62] in this stack alphabet), which gives us a push down system that is congruent to the original statechart. Each input or output event for a given FSM_i in the above decomposition will be pushed into the stack, S_i, of the FSM_i asynchronously (as soon as they arrive), the stack alphabet being completely determined by the corresponding i/o event symbols, because the restricted semantics of Event-clock visibly pushdown automata (ECVPA) is adopted for reasons given earlier. The ECVPAs can then be run asynchronously.

(iii) Non-blocking communication between pairs of ECVPA could be implemented as port structures under prioritised synchronous composition with respective automata, event alphabets for the port structures being the intersection of their respective event alphabets. This construction permits us to interpret the port structures as supervisors implementing required non-blocking communication specification between pairs of ECVPA. The controllability of a given local specification language K ($\subseteq L(P^{B-A})$) (untimed) is checked with $L(P^{B-A})$, where P is one of the automata concerned, with priority set A and driven event set B-A. As demonstrated in [57], it is possible to enforce temporal specifications like: "if p holds when a procedure is invoked, then the procedure must return within d time units and q must hold at the return state" on ECVPA. The example below shows how such specifications can be incorporated into a port structure.

Let us consider a temporal specification involving three events, a, b and c:

"c must occur within 50 time units of a and b *must* occur within 2 time units of the last a."

The ECVPA, M of Fig. 1 below formalises this specification.

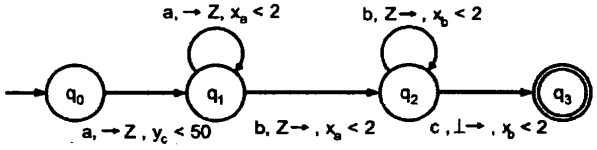

Figure1 - ECVPA, M.

Here a is a push and b and c are pops. Z is a stack symbol. The ECVPA uses two event-recording clocks x_a and x_b, and an event-predicting clock y_c. The transitions of M are given as follows:

- Push: $(q_0, a, y_c<50, q_1, Z)$, $(q_1, a, x_a<2, q_1, Z)$.
- Pop: $(q_1, b, x_a<2, Z, q_2)$, $(q_2, b, x_b<2, Z, q_2)$, $(q_2, c, x_b<2, \perp, q_3)$:

the following conventions are used to represent edges: for instance, a push-transition (q_i, a, ϕ, q_j, Z) is labelled as $a, \rightarrow Z, \phi$; a pop-transition (q_i, b, ϕ, Z, q_j) is labelled as $b, Z \rightarrow, \phi$.

The clock constraint $y_c<50$ that is associated with the edge from q_0 to q_1 ensures that c occurs within 50 time units of the first a. The constraint $x_a<2$ that is associated with the edge from q_1 to q_2 makes sure that the first b occurs within 2 time units of the last a.

The automaton M accepts the non-regular timed language: $L(M) = \{(\bar{\alpha}, \bar{t}) \mid \bar{\alpha} = a^n b^n c,\ n \in N^+,\ t_{i+1} < t_{i+2}, \forall (1 \leq i \leq 2n);\ t_{2n+1} - t_1 < 50\}$. This timed language, however, cannot be accepted by any timed automaton [57].

(iv) For additional supervisory specifications, local supervisor design passes through C&D co-observability of Yoo and Lafortune [34] which also provides a construction algorithm for non-blocking decentralised supervisory control.

(v) For performance specifications given in temporal logics that may be easier to model as Buchi automata, we envisage a suitable adaptation of ECVPA to accommodate the standard testing procedures used with them. In these procedures Buchiaautomata is constructed, which is then complemented using the algorithm in [42]. The language of this complemented automaton should not be accepted by the controlled plant model. This test can be performed by simply verifying that the intersection of languages generated by them is the null set.

A Buchi automaton [48] is a non-deterministic finite automaton on denumerable words. Formally, it is a tuple$A=\{\Sigma,S,\rho,s_o, F)$, where Σ is an alphabet, S is a set of states, ρ:S x$\Sigma\rightarrow 2^s$ is a non-deterministic transition function,$s_0\in S$ is a starting state, and $F\subset S$ is a set of designated states. A run of A over a denumerable word$w=a_1a_2...$, *is a sequence* $s_o,s_1,...$, where $s_i\in\rho(s_{t-1},a_i)$, *for all* $i\geq l$. *A run* $s_o,s_1,...$ *is accepting if there* is some designated state that repeats infinitely often, i.e., for some s$\in$F there are infinitely many i's such that s_i =s. The denumerable word w is accepted by A if there is an accepting run of A over w. The set of denumerable words accepted by A is denoted $L(A)$. Since the difference here is in acceptance condition of a given language we can expect to find suitable adaptations for implementation in the context of ECVPA.

We illustrate our formal verification process though its application to a high-speed railway signalling/ interlocking system.

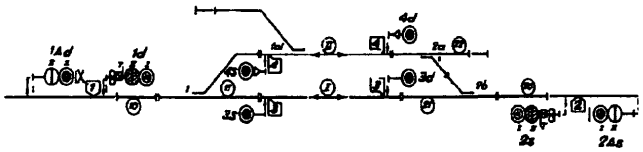

Figure 2 - Typical Interlocking system.

Figure 2 gives a typical layout as presented in [43]. This example (a single track line station) is constituted by eight allowed routes, two switches, eight signals, six track circuits and two automatic blocks. Linesegments represent track segments in the station; some of them have track circuits, that is, sensors that detect the presence of a train, which are numbered inside circles; joints between segments represent switches. The other circular symbols represent signals of various types. Numbered labels are attached to each important part of a route. Interlocking rules being the core of the system, their correctness is the main objective to be addressed. The rules aim at allowing only the safe combinations of switch positions, signals and trains in a station in order to avoid collisions. The signal indications, handled by the interlocking system, govern the correct use of the routes, authorising them ovement of trains. The rules usually enforce a predefined sequence of actions: issuing a route request command usually triggers a check that all the track elements involved in the route are free. In this case, commands are issued for the positioning of switches for that route and for locking the track elements. This phase may be followed by a global centralized control over the correct state of the commanded elements, after which the route is locked and signal indications for the route are set. A route can be set free only if all switches on it are in the correct position, and no trains are present. The signals can be set to green only if the route in front of them is set to free. The above set of rules expresses two examples of generic principles that hold for every interlocking system. Actually, the precise and complete set of such rules depends on the particular station or railway yard, and also on national policies traditionally established by railway companies or regulatory boards. The development of computer controlled Railway Interlocking Systems has seen an increasing interest in the use of Formal Methods, due to their ability to precisely specify the logical rules that guarantee the safe establishment of routes for trains through a railway yard.

The following simplified example of a railway station (Figure 3) reported in [44] will serve in presenting the proposed verification technology. We modify this scenario to suit a situation where the control of railway interlocks is decentralized in such a way that a train communicates with a forthcoming station and initiates the securing of the route before it is reached. This communication could be carried out via radio transmission. Thus the system is separated into an *Enable_Entry* part within the train and a *Yard_Control* part at the yard.

All devices, including track TX and TY, switch units PI and P2 , signal units S 1, S2 and SSX in Figure 3, are also autonomous and decentralized. They communicate with each other directly without a centralized interlocking computer. All these devices can be modeled as local plants with their own control logic: Signal units, Switch Units and Track Units.

The Statechart for Interlocking system is given in Figure 6 with only sufficient detail to illustrate the example. In order to demonstrate our verification method we need to extract out a sub-problem of manageable proportions to

develop in full detail. We shall do this around a *control-point*.

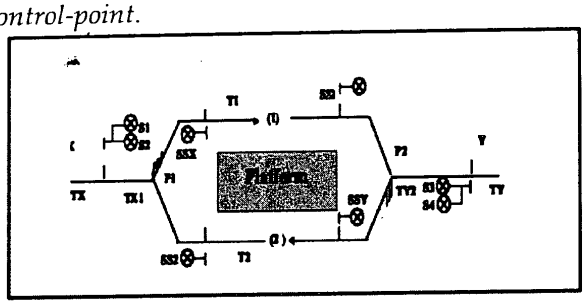

Figure 3 -Basic railway interlocking example.

When a train comes close to the platform, at the *activation point* a route will be requested to allow the train to enter the station safely. Thus, the signal indicating the requested route starts to poll and other devices are informed for relating to the requested route. When the train comes onto track TX and plans to enter platform 1, the signal S1 has to become Green, meaning that the train has been permitted to enter platform 1. Signal S1 becomes green only when all necessary conditions are satisfied.

A control point (CP) is a safety measure which is situated directly in front of each junction (point). The train control models control points in an internal representation and in order to pass a control point the train has to delete this representation. Naturally this should only be done once the route has been secured.

Control points have a major impact on the speed control, in particular on the computation of the maximal nominal speed. The maximal nominal speed is influenced by the train's maximal speed, the maximum speed allowed on the current part of the track and the presence of control points. Its development over time is commonly pictured by a curve (see figure 7). Here the situation that a train approaches a control point is shown. From the activation point on the maximal nominal speed steadily decreases until it reaches zero right in front of the control point. This is the default behavior for all control points; once the hazard the control point represents—e.g. an unsecured routing—is eliminated and its representation deleted, the maximal nominal speed will be set back to the regular speed for this track and train. Note that the maximal nominal speed is an ideal value which in practice is often approximated in discrete steps resulting in an *enforced* maximal nominal speed. Figure 7 shows an example for such an approximation. The *actual speed* of the train is determined by the minimum of enforced maximal nominal speed and the speed set by the driver [45].

When the train passes the *activation point* on its way to the station it generates a Route_Request (R_Req) which is read by Yard_ctrl. The Switch will eventually switch the route setting and the Signal lights changed accordingly. Also the Train. Enable_Entry_ctrl should independently receive a Route_Status signal in response to its Stat_Req. This is the normal behaviour. In an error situation, the Route_Status signal might not arrive before the timer is out (tm1 = 1). In the latter case, control passes to Speed_control and this becomes the Activation point for the enforced maximal nominal speed curve. Train will then be automatically stopped before the Control point. The train can only continue after a fault recovery procedure or once the driver manually confirms that the route permission has been granted.

A simple design for Speed_Control statechart is given in Figure 8 [45].

An algorithm from [45] for computing the current enforced nominal speed as a function of the Odometer reading is given in Figure 9 below.

The formal verification procedure described in steps (i) to (v) will now be illustrated on these statecharts.

Step (i): The construction of ECVPA tree for the *Interlock* statechart (Fig. 6)is as shown in Figure 4, with the only difference that internal variables are added to the links.

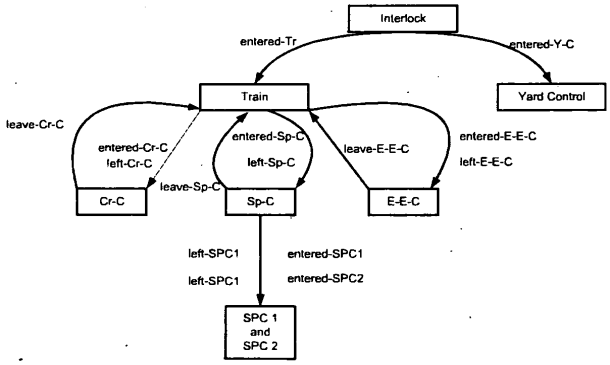

Figure 4 - Part of the ECVPA tree for Interlock.

Transitions in the ECVPA tree can be viewed as events generated by clients (always identified as states in the statechart) addressed to servers (of the type defined by Harel and Geryin [63]) that a realways constrained to be from the sub-states of the respective clients at the immediately next level. The respective servers are then forced to use these events in triggers on their transitions (these triggers can be complex conditions).

Step (ii): Figure 10 gives the $ECVPA_{sc}$ for Speed_control substate in the FSM tree of Step (i).

Let a = (Rt = 0), b = Entered_Speed_Ctrl, c = RReq, d = Left_Speed_Ctrl, e = Activation_point, f = entered_SPC1, g = entered_SPC2, h = left_SPC1, i =

left_SPC2, j = [NORMAL_SPEED < ODATA_SPEED], k = RELEASED_MAN, m = ¬(NORMAL_SPEED < ODATA_SPEED>), o = ¬(ODATA_SPEED ≤ 0), p = ODATA_SPEED ≤ 0, q = NOMINAL_SPEED ≤ 0,$s1$ = BRAKE, $s2$ = STPPED, $s3 = m \wedge o$, $s4 = p \wedge q$

$\Sigma^{sc}_c = \{a, b, d, e\}$, $\Sigma^{spc2}_c = \{g, i, j, k, s3, s4\}$

$\Sigma^{sc}_r = \{c\}$, $\Sigma^{spc2}_r = \{c, s1, s2\}$

$\Gamma = \{\gamma_a, \gamma_b, \gamma_d, \gamma_e, \gamma_c, \gamma_d, \gamma_e, \gamma_f, \gamma_g, \gamma_h, \gamma_i, \gamma_j, \gamma_k, \gamma_m, \gamma_o, \gamma_p, \gamma_q, \gamma_{s1}, \gamma_{s2}, \gamma_{s3}, \gamma_{s4}\}$, $\gamma_i = \gamma_f = \gamma_g = \gamma_h = \perp$

All virtual variables as defined above are handled solely by the port structures between individual automata. Figure 11 is the $ECVPA_{spc2}$ for SPC2.

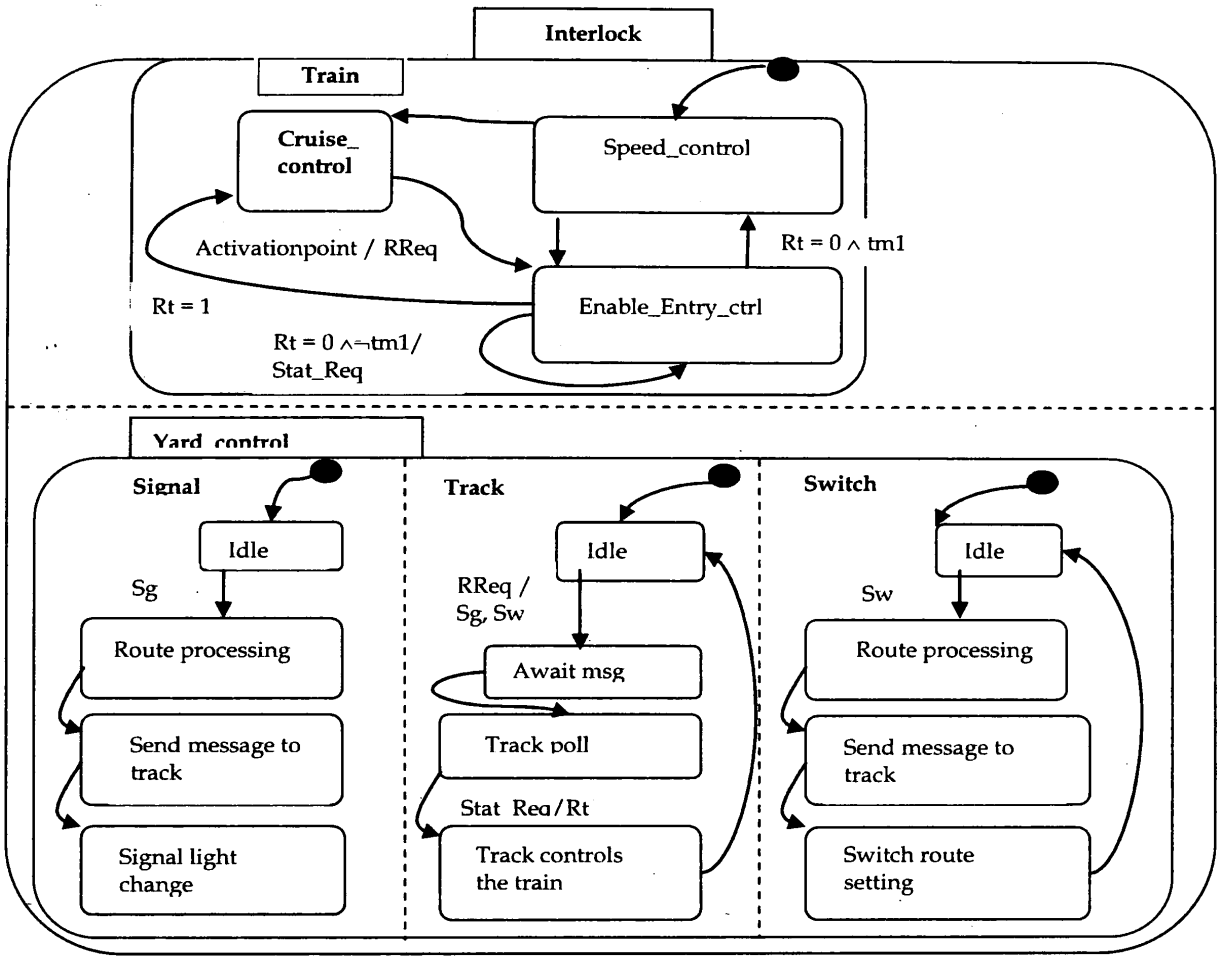

Figure 6 - Statechart for Interlocking system.

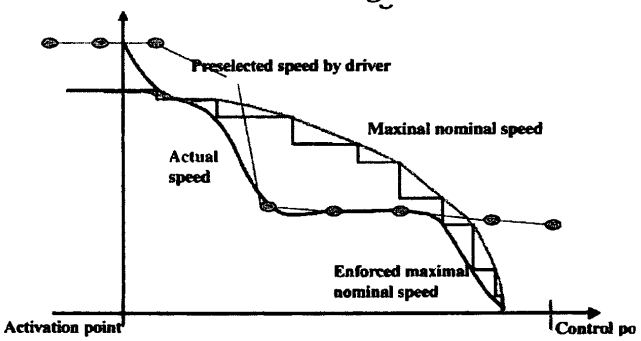

Figure 7 - Speed curves for the train [45].

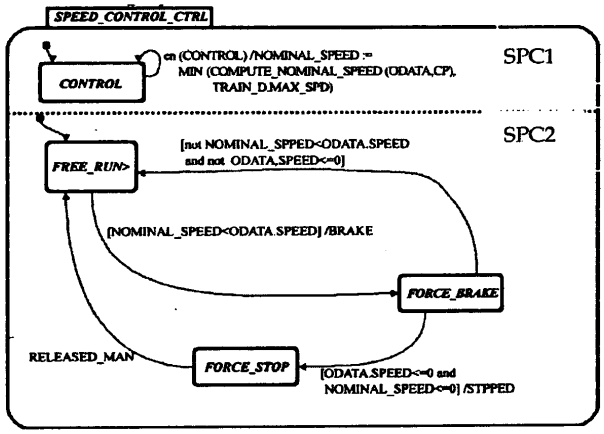

Figure 8 - Statechart for speed control [45].

```
if CP.ALREADY_REGARDED = true then
  RET := 40
else
  DIST := CP.POS – TRAIN.POS;
  if DIST <= 5 then RET := 0
    else if DIST <= 30 then RET := 10
      else if DIST <= 50 then RET := 20
        else if DIST <= 80 then RET := 30
        end if
      end if
    end if
  end if;
return(RET)
```

Figurer 9 - Algorithm for computing the current enforced nominal speed [45].

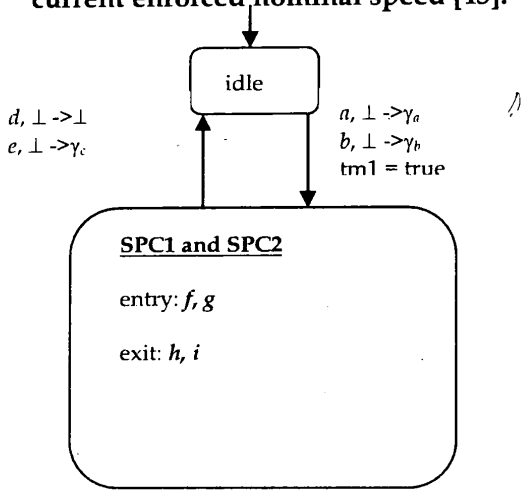

Figure 10 - ECVPA$_{sc}$ for Speed_Ctrl

Step (iii): Construction of communicating language generators (LG's) and port structures (PS'):

Convert the FSM underlying each ECVPA into a language generator, with a suitable definition of controlled and uncontrolled events, to represent plant modules to be used in supervisor synthesis based on supervisory control theory [1], [2].

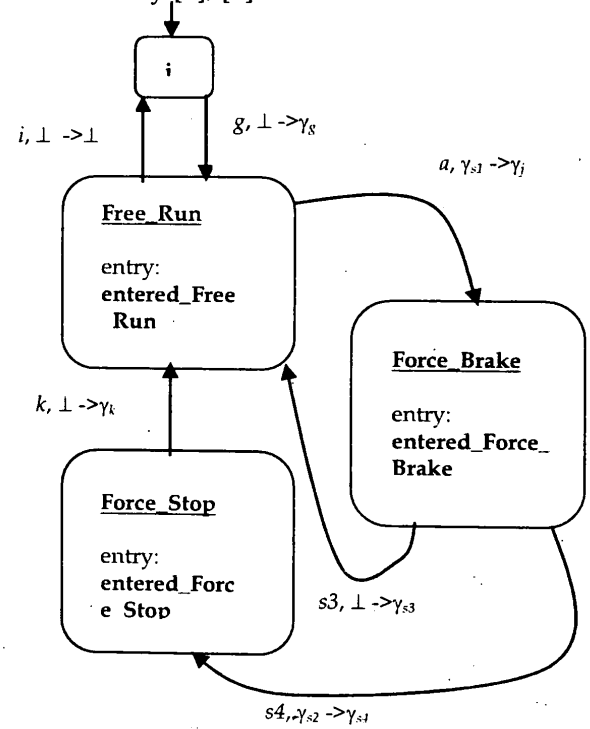

Figure 11- ECVPA$_{spc2}$ for SPC2.

Use the tree of interconnected FSMs above to identify event/action communication between pairs of language generators. Define port structures between pairs of language generators using these events and interpret these port structures as supervisor automata implementing specifications on event communication.

The super-state SPC2 (Fig. 8) with three sub-states Free_Run, Force_Stop and Force_Brake results in the following sub-tree of LG's:

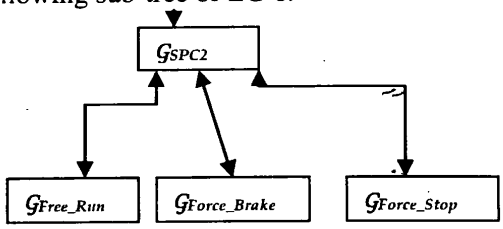

Figure 12 - Statechart decomposition.

The port alphabet for communication between G_{SPC2} and G_{Force_Brake} is then $\Sigma^P = \Sigma^{SPC2} \cap \Sigma^{Force\_Brake}$ = {*entered_Force_Brake, left_Force_Brake, BRAKE, STPPED*}, the intersection of their respective event alphabets. Here $\Sigma^{SPC2} = \Sigma^{spc2}{}_c \cup \Sigma^{spc2}{}_r \cup \Sigma^{spc2}{}_l$ as defined in Ch.4.For clarity of presentation we will continue to use *BRAKE* and *STPPED* for *s1* and *s2*, respectively, in the rest of the paper. The port automaton, ζ_P, is constructed by masking the transitions corresponding to events **not** in Σ^P from G_{SPC2} and G_{Force_Brake} and taking the union of the two resulting LG's. Controllability check under prioritised-synchronous composition between ζ_P and G_{SPC2} and test for non-blocking behaviour succeed immediately. Same is true between ζ_P and G_{Force_Brake}. Broadcast communication of events generated on state entry and state transition (*BRAKE* and *STPPED*, for instance) is implemented by pushing them into the respective stacks. They can be popped by any transition that can consume them. This is the asynchronous mode. In synchronous mode these variables are represented by FIFO queues of update values. The value at the head is consumed after a synchronous read by all receptive transitions. Port structures for these communications are built and tested in an analogous manner. Non-blocking simulation run of the simulink-statechart model validated these results. We do not give finer details of the controllability check here because they were presented in [17], albeit before the introduction of internal event stack/queue, as appropriately. We intend to include these in a separate communication.

Similarly, we are making an effort to incorporate C&D co-observability checks for

local supervisory specifications and the corresponding infimal controllable and co-observable super-languages, because they seem to be useful in highly decentralised systems like high-speed railway control. Difficulties to be overcome here are related to the undecidability of non-blocking distributed supervisor existence in the distributed plant case.

Now we will demonstrate how performance and safety specifications given in **temporal logics** could be incorporated following the method given in section 4(v) above:
One can think of two kinds of rules for this system. Some rules specify the way the system must respond to some particular input condition. These rules can be written using the following formula:
G(<input>→ F(<target> W¬<input>)).
It says that the target station must be reached if the input happens, and the target station will stay at that target state until the input is reset.

Others rules specify the way the system must hold independently of the state of the inputs. These rules can be considered propositional invariants of the system. Such rules can have the following form:**G(<invariant>)** .
For instance, when the track is free and signal turns green from red, it enables the train passage through its track. Thus, for safety reasons, it is necessary that both tracks must be locked in order to avoid any accident that may cause train derailment. This rule is input-independent and may be stated by the invariant (track TI for example):
G(TI=free & SSI=green → TI=locked).
Opposite signals must never be on at the same time.
Consider track T2 and signals SSY and SS2 as example, and the corresponding rule could be:
G ¬(SSY=green & SS2=green).
Several temporal logics have been shown to have exactly the expressive power of **Buchi**automata; in other words, the class of sets of sequences described by those logics coincides with the class ofω-regular languages [42]. One method to decide satisfiability for these logics is to build a Buchi automaton that accepts exactly the strings satisfying the formula. Since these logics are closed under negation, building this automaton involves complementing Buchi automata. That is, given a Buchi automaton A, one has to find a Buchi automaton $\bar{A}$such that $L(\bar{A}) = \Sigma^{\omega} - L(A)$, where $L(A)$ denotes the language accepted by A. Now, one direct way of verifying that the specification is satisfied is to show that $L(\bar{A})$ is not accepted by the controlled plant, S/G: i.e.,$L(S/G) \cap L(\bar{A}) = \Phi$. In [48] Ramadge gives polynomial time algorithms for computing the minimally restrictive supervisor satisfying the specification.
For all specifications with local scope, controllability of the language specified by each supervisor automaton (representing a port structure) can be checked separately for each of the corresponding plant generators. Then the non-conflicting nature of each supervisor automaton (representing a port structure) can be verified separately with each of the corresponding plant generators, with no guarantee of globally non-blocking behaviour
When this process of modular verification is completed for each pair of communicating FSM, the statechart can be considered as verified for the same properties, particularly when each port structure attached to a given module can be considered as the generator of a closed language (local supervisory specification). We base this conclusion on (2.8) given earlier. This is a requirement that can be enforced on port construction. Under this constraint, the combined action of several port structures on any local plant module would be controllable and non-blocking. The network of plant modules that result from the proposed decomposition of statechart specification would then remain non-blocking.

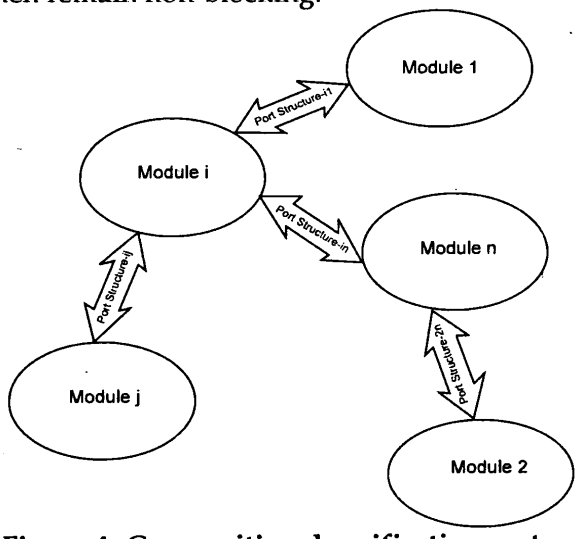

Figure 4 -Compositional verification system based on port structures.

Subsequent changes in any particular FSM corresponding to a change in hardware/software can be verified locally.
In [17] we investigated verification of non-blocking and correct interaction among FSMs resulting from a D-H like distributed implementation of statecharts. In the present communication the analysis has been extended beyond the level of verification currently realised in (symbolic) model checkers.

6. Conclusion

The problem addressed in this research is the compositional (formal) verification of control specifications given using statecharts. The applicability of a compositional safe by design strategy being developed by the author for complex reactive systems is demonstrated. The proposed model is a decomposition of the statechart specification into a set of communicating pushdown automata (of a restricted kind). When this process of modular verification is completed for each pair of communicating event-clock-visually-pushdown automata (ECVPA), the statechart can be considered as verified for the same properties. The proposed new synthesis successfully confronts the modular verification/implementation problem for statecharts. A regular language (hence, finite state automata) based version of this approach was reported in [16] and [17]. Application development on this version has been carried out with collaborators on a purpose built elevator test rig and reported in [58], [59] and [60]. In the current paper the method has been extended to include internal event processing in real-time contexts by elevating the method to computationally more powerful context-free languages (and hence, pushdown automata) and facilitating the integration of supervisory specifications of RW type and general correctness properties expressed in temporal logic. The approach is demonstrated on a high-speed railway signalling application.

A complete analysis and design process for the target system has been demonstrated using realistic problem scenarios. Subsequent changes in any particular module (FCVPA) corresponding to a change in hardware/software can be verified locally.

The collaborative axis of this research programme using the regular language based approach continues, where a PLC based distributed implementation of the methodology using IEC 61499standard function blocks and an FPGA based implementation have been carried out for controlling the elevator system and published, authored with collaborators [57, 58, 59]. This axis of research should produce results on redundancy based design, hardware-in-the-loop level testing and model-based design in Matlab/Simulink/Stateflow environment and will be published with collaborators.

References

1. Ramadge P.J. and Wonham W.M., "Supervisory Control of a Class of Discrete Event Systems," in SIAM J. Control and Optimization 25 (1987), no. 1, 206-230.
2. Ramadge P.J. and Wonham W.M., "The Control of Discrete Event Systems," in Proceedings of the IEEE 77 (1989), no. 1, 81-98.
3. Sampath M., Lafortune S., and Teneketzis D., "Active Diagnosis of Discrete Event Systems," in IEEE Transactions on Automatic Control 43 (1998), no. 7.
4. Harel D., "Statecharts: A Visual Formalism for Complex Systems," in Sci. Comput. Prog.vol. 8, pp. 231-274, 1987. (Preliminary version appeared in CS84-05, Weizmann Institute of Science, Rehovot, Israel, Feb. 1984.).
5. Bhaduri P. and Ramesh S., "Model Checking of Statechart Models Survey and Research Directions," Computing Research Repository, arXiv:cs/0407038 [cs.SE], 2004.
6. Chan W., Anderson R.J., Beame P., Burns S., Modugno F., Notkin D., and Reese J.D., "Model Checking Large Software Specifications," in *IEEE Transactions on Software Engineering*, 24(7), 1998.
7. Drusinsky D., and Harel D., "Using Statecharts for Hardware Description and Synthesis," in IEEE Transactions on Computer-Aided Design. 8(7). July1989.
8. Akesson K., Flordal H. and Fabian M., "Exploiting Modularity for Synthesis and Verification of Supervisors," in *Proc. of the IFAC*, July, 2002.
9. Flordal H., Fabian M., Kesson K. and Hellgren A., "Controllability Revisited: a Generalization for The Modular Approach," *IFAC*, 2004.
10. Golaszewski, C. H. and Ramadge P.J., "Control of Discrete Event Processes with Forced Events," in Proceedings of 26th IEEE Conference on Decision and Control, pages 247-251, Los Angeles, CA, 1987.
11. Shayman M.A. and Kumar R., "Supervisory Control of Non-deterministic Systems with Driven Events via Prioritized Synchronization and Trajectory Models," inSIAM Journal of Control and Optimization, 33(2):469-497, March 1995.
12. Zhou C. and Kumar R., "Prioritized Synchronization under Mask for Interaction/Control of Partially Observed Discrete Event Systems," submitted, 2006.

13. Endsley E.W., Lucas M.R. and Tilbury D.W., "Software Tools for Verification of Modular FSM Based Logic Control for Use in Reconfigurable Machining Systems," Japan-USA Symposium on Flexible Automation, Ann Arbor, Michigan, USA, 23-26 July, 2000.
14. Huang Y., Jeng M. and Hsu C., "Modelling a Discrete Event System Using Statecharts," in Proceedings of IEEE International Conference on Networking, Sensing and Control, Taipei, Taiwan, March 21-23, 2004.
15. Lam V.S.W. and Padget Julian, "Symbolic Model Checking of UML Statechart Diagrams with an Integrated Approach," in Proc. of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04). 2004.
16. Dewasurendra S.D., "Statecharts for Reconfigurable Modular Control of Complex Reactive Systems: need for Formal Verification and Associated Problems," in Proceedings of ICIIS 2006, Peradeniya, Sri Lanka, 09-11 August, 2006.
17. Dewasurendra S.D., "Statecharts for Reconfigurable Modular Control of Complex Reactive Systems: A New Formal Verification Methodology," in Proceedings of ICIIS 2006, Peradeniya, Sri Lanka, 09-11 August, 2006.
18. Cao X. and Ho Y., "Models of Discrete Event Dynamic Systems", IEEE Control Systems Magazine (1990), 69-76.
19. Guia A. and DiCesare F., "Blocking and Controllability of Petri nets in Supervisory Control", IEEE Transactions on Automatic Control 39 (1994), no. 4, 818-824.
20. Ozveren C. and Willsky A.S., "Observability of Discrete Event Systems", IEEE Transactions on Automatic Control 35 (1990), no. 7.
21. Cieslak R., Desclaux C., Fawaz A.S., and Varaiya P., "Supervisory Control of Discrete-Event Processes with Partial Observations", IEEE Transactions on Au- tomatic Control 33 (1988), no. 3, 249--260.
22. Kumar R. and Garg V.K., "Modeling and Control of Logical Discrete Event Systems", Kluwer Academic Publishers, 1995.
23. Ramadge P.J. and Wonham W.M., "Modular Feedback Logic for Discrete Event Systems", SIAM J. Control and Optimization 25 (1987), no. 5, 1202--1218.
24. Lin F. and Mortazavian H., "A Normality Theorem for Decentralized Control of Discrete-Event systems", IEEE Transactions on Automatic Control 39 (1994), no. 5, 1089--1093.
25. Heymann M. and Lin F., "Discrete Event Control of Non-Deterministic Systems", Proceedings of the 35th IEEE CDC, December 1996, pp. 4445--4450.
26. Brandin B. and Wonham W.M., "Supervisory Control of Timed Discrete-Event Systems", IEEE Transactions on Automatic Control 39 (1994), no. 2, 329--341.
27. Li Y. and Wonham W.M., "Concurrent Vector Discrete-Event Systems", IEEE Transactions on Automatic Control 40 (1995), no. 4, 628--637.
28. Sampath M., Lafortune S., and Teneketzis D, "Active Diagnosis of Discrete Event Systems", IEEE Transactions on Automatic Control 43 (1998), no. 7.
29. Wong K.C. and Wonham W.M., "Hierarchical Control of Discrete Event Systems", Discrete Event Dynamical Systems 6 (1996), 241--273.
30. David Alexandre, Moller M. Oliver and. Yi Wang. "Formal Verification of UML Statecharts with Real-Time Extensions", in Proc. of Fundamental Approaches to Software Engineering (FASE'2002) and Vol. 2306 of LNCS. 2002.
31. Henzinger Thomas. A., Nicollin Xavier, Sifakis Joseph, and Yovine Sergio "Symbolic Model Checking for Real-Time Systems", *Information and Computation*, 111(2):193-244, 1994.
32. Lam, Vitus S.W. and Padget Julian "Symbolic Model Checking of UML Statechart Diagrams with an Integrated Approach", Proc. of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04). 2004.
33. Lin, F. and Wonham, W.M. Decentralized Supervisory Control of Discrete-Event Systems. Information Sciences. 44, 1988. 199-224.
34. Yoo T.-S. and Lafortune S., "Decentralized Supervisory Control with Conditional Decisions: Supervisor Existence", IEEE Trans. Automat. Contr., vol. 49, no. 11, pp. 1886-1903, 2004.
35. Alur R. and Yannakakis M., "Model Checking of Hierarchical State Machines," in *ACM Transactions on Programining Languages and Systems*, 23(3):273–303, May 2001.
36. Rhapsody, Model-driven development with UML2.0, SysML and Beyond. http://www.ilogix.com.
37. Lam V. S.W. and Padget Julian, "Symbolic Model Checking of UML Statechart Diagrams with an Integrated Approach," in Proc. of the

11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04). 2004.

38. Abdelwahed S. and Wonham W. M., "Supervisory Control of Interacting Discrete Event Systems", in Proc. of the 41st IEEE Conference on Decision and Control Las Vegas, Nevada USA, December 2002.

39. Willner Y. and Heymann M., Supervisory Control of Concurrent Discrete Event Systems. *Int. Journal of Control*, 54(5):1143-1169, 1991.

40. Rudie K. and Wonham W. M., "Think Globally, Act Locally: Decentralized Supervisory Control", IEEE Trans. Automat. Contr., vol. 37, no. 11, pp.. 1692-1703, 1992.

41. Tiwari A., "Formal Semantics and Analysis Methods for Simulink Stateflow Models", Unpublished report.

42. Sistla A. P., Vardi M. Y. and Wolper P., "The Complementation Problem for Buchi Automata with Applications to Temporal Logic", Theoretical Comput. Sci., Vol. 49, pp. 217-237, 1987.

43. Banci M., Fantechi A. and Gnesi S., "Experimenting with Diversity in the Model Driven Development of a Railway Signaling System", Proceedings of International Conf. on Computer applications and System Modelling (ICCASM), 2010.

44. W. Ma and Hei X., " A Approach for Design and Formal Verification of Safety-Critical Software",

45. Damm W. and Klose J., "Verification of Radio-Based Signalling System using the STATEMATE Verification Environment", Formal Methods in System Design, Vol. 19, pp. 121-141, 2001.

46. Griffin C., "A Note on Deciding Controllability in Pushdown Systems", IEEE Trans. Automat. Contr., Vol. 51, No. 2, pp. 334-337, 2006.

47. Berry G., Berry G., The Foundations of Esterel. In Plotkin G., Stirling C., and Tofte M., editors, Proof, Language, and Interaction: Essays in Honourof Robin Milner. MIT Press, 2000.

48. Ramadge P. J. G., "Some Tractable Supervisory Control Problems for Discrete-Event Systems Modeled by Buchi Automata", IEEE Trans. Automat. Contr., Vol. 34, No. 1, pp. 10-19, 1989.

49. Holzmann G. J., *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004. ISBN 0-321-22862-6.

50. http://nusmv.fbk.eu

51. Holger Giese and Sven Burmester. *Real-Time Statechart Semantics*. Technical Report tr-ri-03-239, Lehrstuhl für Softwaretechnik, Universität Paderborn, Paderborn, Germany, 6 2003.

52. Alur R. and Dill D.L., A Theory of Timed Automata. Theoretical Computer Science,126:183{235, 1994.

53. Huizing C., Gerth R. and Roever W P. de, Modelling Statecharts Behaviour in a fully Abstract Way, Computer Science Notes, F.A.J. van Neerven, Ed.,Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands,1988.

54. Hopcroft J. E. and Ullman J. D., Introduction to Automata Theory, Languages and Computation, 1979 edition, Addition-Wesley, 1979.

55. Alur R. and Madhusudan P., Visibly pushdown languages, in STOC 04, ACM, 2004, pp. 202-211.

56. Alur R., Fix, L. and Henzinger, T.A. *Event-Clock Automata: A Determinizable Class of Timed Automata.* Theoretical Computer Science, 211, 253-273 (1999).

57. Nguyen *Tang*, Mizuhito Ogawa, Event-Clock Visibly Pushdown Automata, SOFSEM '09: Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 24-30, 2009.

58. Dewasurendra S.D., Vidanapathirana A. and Abeyaratne S.G., "Development of an Elevator Prototype for Multi-Disciplinary Research in Complex Reactive Systems," Annual Transactions, Volume I - Part B, The Institution of Engineers, Sri Lanka, 2011.

59. Jayawardana H.P.A.P., Amarasekara H.W.K.M., Peelikumbura P.T.S., Jayathilaka W.A.K.C., Abeyaratne S.G., Dewasurendra S.D. (2011) "Design and Implementation of a Statechart Based Reconfigurable Elevator Controller," Sixth International Conference on Industrial & Information Systems (ICIIS 2011), 16th to 19th August, Kandy, Sri Lanka, IEEE, IEEE Xplore.

60. Vidanapathirana A.C., Dewasurendra S.D., Abeyratne S.G., (2011) "Statechart Based Modeling and Controller Implementation of Complex Reactive Systems," Sixth International Conference on Industrial & Information Systems (ICIIS 2011), 16th to 19th August, Kandy, Sri Lanka, IEEE, IEEE Xplore.

61. Damm W., Josko B., Hungar H., and Pnueli A., A Compositional Real-Time Semantics of STATEMATE Designs. In Roever W.-P. de, Langmaack H., and Pnueli A., editors, Proc.

COMPOS 97, Lecture Notes in Computer Science 1536. Springer-Verlag, 1997.

62. Eshuis R. and Wieringa R., Requirements-Level Semantics for UMLstatecharts. InS.F. Smith and Talcott C.L., editors, Formal Methods for Open Object-Based Distributed Systems IV (FMOODS 2000), pages 121 140. Kluwer Academic Publishers. Kluwer Academic Publishers, 2000.

63. Harel D. and Gery E., Executable Object Modelling with statecharts. In IEEE COMPUTER, July 1997.