Semi-Formal and Formal Models Applied to Flexible Manufacturing Systems*

Andrea Matta[1], Carlo A. Furia[2], and Matteo Rossi[2]

[1] Dipartimento di Meccanica, Politecnico di Milano
9, Via Bonardi, 20133 Milano, Italy
andrea.matta@polimi.it
[2] Dipartimento di Elettronica e Informazione, Politecnico di Milano
32, Piazza Leonardo da Vinci, 20133 Milano, Italy
{furia,rossi}@elet.polimi.it

Abstract. Flexible Manufacturing Systems (FMSs) are adopted to process different goods in different mix ratios allowing firms to react quickly and efficiently to changes in products and production targets (e.g. volumes, etc.). Due to their high costs, FMSs require careful design, and their performance must be precisely evaluated before final deployment. To support and guide the design phase, this paper presents a UML semi-formal model for FMSs that captures the most prominent aspects of these systems. For a deeper analysis, two refinements could then be derived from the UML intermediate description: simulation components for "empirical" analysis, and an abstract formal model that is suitable for formal verification. In this paper we focus on the latter, based on the TRIO temporal logic. In particular, we hint at a methodology to derive TRIO representations from the corresponding UML descriptions, and apply it to the case of FMSs. A subset of the resulting formal model is then used to verify, through logic deduction, a simple property of the FMS.

Keywords: Formal models, UML, temporal logic, flexible manufacturing systems.

1 Introduction

Flexible Manufacturing Systems (FMSs) are widely used in shop floors to produce a large set of product families in small/medium volumes. In FMSs, the high flexibility of machines allows manufacturing different products in different mix ratios, thus providing firms the ability to react quickly and efficiently to changes in products, volumes and mix ratios. However flexibility has a cost, and the capital investment sustained by firms to acquire such types of manufacturing systems is generally very high. Therefore, in the development of a FMS, particular attention must be paid to the design phase, where the performance of the system has to be accurately evaluated so that the most appropriate resources (those that optimize the processes of the FMS) are selected since the early stages of the development process [4].

* Work supported by the FIRB project: "Software frameworks and technologies for the development and maintenance of open-source distributed simulation code, oriented to the manufacturing field." Contract code: RBNE013SWE.

Discrete event simulation is generally used to estimate in detail the performance of FMSs. Starting from a description of the FMS, which can consist of a simple natural-language representation of the system or a more formal description, a simulation model is built, validated and run to numerically obtain the system performance. The development phase of a simulation model is time-consuming and depends on the complexity of the system and the (mostly informal) way in which it is described. In addition, simulation experiments need long computations to obtain a deep knowledge of the system and of its structural properties. Current FMS design techniques rely heavily on simulation models [4], and the specifications from which these models are built are still mostly written in natural language. This practice requires that the implementors of the simulator are also domain experts who can precisely understand the initial description.

In this paper we address these limitations in two ways. As a first contribution, a semi-formal UML [5] model for describing FMSs is developed. The UML description is used as the conceptual model of the system during the building of the simulator, thus improving the development phase. In addition, the UML representation can be used as a *lingua franca* between the specifiers of the FMS and the implementors of the simulation model, who now need not be domain experts in order to build the simulator (although a certain understanding of the application cannot be entirely eliminated).

Despite its popularity in the software-development world, standard UML is inadequate to carry out precise analysis of systems for its lack of a well-defined semantics. To fill this gap, two approaches can be followed: one could try to give formal semantics to the various UML diagrams, or UML diagrams could be translated in an existing formal notation. Here we follow the latter approach: as a second contribution, this paper hints at a simple methodology to derive formal descriptions expressed in the TRIO temporal logic [2] from semi-formal UML diagrams. TRIO is a powerful formal language that supports a variety of verification techniques (test case generation, model checking, theorem proving). In this work, the aforementioned methodological steps are applied to the UML-based FMS representation to derive a TRIO-based formal description. This TRIO model supports and eases the design phase of FMSs through formal verification. In particular, it allows one to perform theorem-proving-based analysis on manufacturing systems, which can spare FMS developers running expensive simulations.

The paper is structured as follows: Section 2 introduces FMSs; Section 3 presents the UML model; Section 4 sketches the methodological steps to derive TRIO descriptions from UML ones; Section 5 shows an example of formal analysis; finally, Section 6 draws conclusions and outlines possible future work.

2 Flexible Manufacturing Systems

FMSs are production systems composed of computer numerically controlled (CNC) machining centers that process prismatic metal components. A process cycle defining all the technological information (e.g. type of operations, tools, feed movements, working speeds, etc.) is available for each product so that the system has the whole knowledge for transforming *raw parts*, the state in which a piece enters into the system, into *finished parts*, the state in which a piece has completed the process cycle. The main components of FMS are described below.

CNC machines perform the operations on raw parts. A machining operation consists in the removal of material from the raw parts with a tool, fixed in the machine. The machines are CNC in that their movements during the machining operations are locally controlled by a computer. Machines can differ in size, power, speed and number of controlled axes.

Pallets are the hardware standard physical interfaces between the system components and the parts. Parts are clamped on pallets by means of automated fixtures that provide stability during the machining operation. Generally, but not always, fixtures are dedicated to products.

Load/unload stations clamp raw parts onto pallets before they enter the system and remove finished parts after their process cycle has been completed by the machines of the system. Stations can be manned, i.e. an operator accomplishes the task, or unmanned, i.e. a gantry robot accomplishes the task.

Part handling sub-system is the set of devices that move parts through the system. Different mechanical devices are adopted in reality: automated guided vehicles, carriers, conveyors, etc.

Tools perform the cutting operations on raw parts. Since tools are expensive resources their number is limited and as a consequence they are moved through the system when requested by machines.

Tool handling sub-system is the set of devices that move tools through the system. The most frequently adopted solution is a carrier moving on tracks.

Central part buffer is the place where pallets wait for the availability of system resources (i.e. machines, carriers, load/unload stations).

Central tool buffer is the place where tools can be stored when they are not used.

Supervisor is the software that controls resources at the system level by assigning pallets to machines and load/unload stations and by scheduling tool/pallet transports.

Tool room is the place where tools are reconditioned after the end of their life.

Let us now describe the flow of parts in the system. Generally more than one part is loaded on pallets at the load/unload station of the system. The type and the number of parts on the same pallet depend on products and system components. In detail, the number of parts depends on the physical dimensions of parts and machines, while the types of products depend on the technical feasibility of clamping different products with the same fixture. Most times the parts loaded on the same pallet are of the same type, however it is increasing the number of cases in which different part types are loaded on the same pallet. If the loading operation is executed manually by operators, then the corresponding time can be considered a random variable according to some estimated distribution; otherwise, if automated devices (e.g. a gantry robot) perform the operations and no source of uncertainty is present in the task, the assumption of deterministic loading/unloading times holds .

After parts are loaded on pallets, they are managed by the supervisor which decides the path each pallet has to follow to complete the process cycle of all its parts. In order to complete the process cycle, pallets must visit at least one machine; if machines are busy, pallets wait in the central buffer. Each machine has at least two pallet positions: the first one is the *pallet working position*, when the pallet is machined by the tool, while the other positions, known as *pallet waiting positions*, are used to decouple the machine

from the part handling sub-system. Indeed, the pallet in the waiting position waits for the availability of the machine, or of the carrier if the pallet has already been worked. The machine is equipped with a pallet changer to move a pallet from a waiting position to the working position and vice versa; this movement is executed by automated devices and can be considered deterministic since there is no source of variability in the operation. After the pallet has been blocked in the working position and the tools necessary for the operations are available to the machine, the processing operations can be executed. Processing times of machines can reasonably be assumed deterministic. In fact, the trajectory of the tool during the cutting of material is computer numerically controlled and therefore the sources of variability are eventually negligible.

3 UML Description of a FMS

The UML model of a generic FMS is now briefly described. Following the usual structure of UML [5], the proposed model is divided into three areas of major interest: model management area, structural area and dynamic area. Each area represents a particular point of view of the modelled system and is presented in the remainder of this section.

3.1 Model Management Area

The only diagram belonging to this area is the package diagram. It represents how the model is organized by grouping the objects of the system into three main packages and by specifying the dependencies among them. The three main packages of a FMS system are: working sub-system, material handling sub-system and supervision sub-system. Each package represents the homonymic sub-system of real FMSs. The supervision sub-system is divided into two sub-packages: system supervisor and operation manager. The former contains the elements dealing with part and tool management policies of the FMS at the system level. The latter includes all system components providing production management and planning tools. All packages depend on the supervisor, which coordinates and schedules all the activities of machines, transporters and load/unload stations.

3.2 Structural Area

In this area both the physical elements and the abstract concepts belonging to a FMS are described using common object-oriented modeling tools such as classes, associations, generalization, aggregation, composition, etc. The supervisor class diagram, shown in Figure 1, defines the management aspects of the system. The Supervisor is a singleton, i.e. only an instance can be defined, and controls at the system level all the macro resources: machining centers, load/unload stations and part/tool handling sub-systems. A Control unit controls locally a Station (i.e. a machining center, or a load/unload station) or a Handling sub-system (i.e. a part or tool handling sub-system). There are as many Control unit instances as the number of workstations and material handling subsystems in the real system. The Supervisor controls the overall system, by scheduling

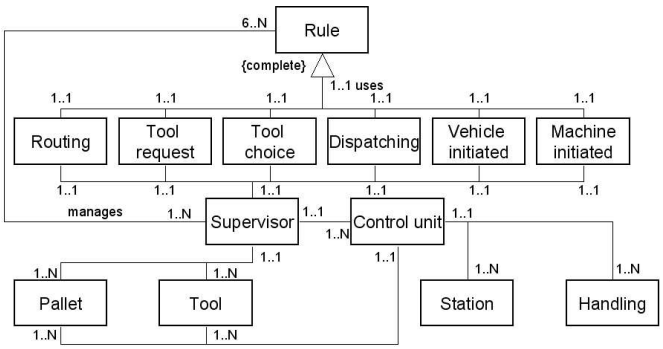

Fig. 1. UML model: supervisor class diagram.

Pallet operations and Pallet/Tools transport by means of the management rules defined in the class Rule, which in turn is made up of six different types of rules.

In the Routing rule class there are rules to select the Pallet path in the system. In practice, different machines can often perform the same types of operations and, therefore, a decision concerning the best machines visiting sequence has to be taken on the basis of the system state. In order to simplify the system management, in most real cases the routing of pallets is decided statically off-line. This is a system level rule used by the Supervisor to schedule the pallet flow. In the Tool request rule class there are rules to select the first tool request to satisfy. Notice that tool requests can be emitted only by machining centers, when they need the tool to process a part, or by the tool room when the tool has finished its life. This is a system level rule used by the Supervisor to schedule the tool flow. In the Tool choice rule class there are rules to select the copy of a tool used to satisfy requests. Typically more than one copy of the same tool is present in real FMSs. This is a system level rule used by the Supervisor to schedule the tool flow. In the Dispatching rule class there are rules to select the first pallet to work (or to load) in the next period. This is a local level rule used by the Control unit of a Station. In the Vehicle and Tool initiated rule classes there are rules to assign a pallet or a tool to a vehicle (only for transport networks with vehicles). The Control unit of a part handling sub-system uses this rule at the local level.

The remainder of the FMS class diagram is not described in this paper for the sake of space. The interested reader may refer to [1] for more details.

3.3 Dynamic Area

This area describes the dynamic behavior of FMSs. Only the dynamics of pallets is presented in this section and the reader can refer to [1] for more details on the tool flow. The most interesting aspects of the pallet flow are the loading/unloading on/from the system, the transport, and the processing at machining centers.

In the loading task of a pallet with raw workpieces, the parts are fixed on a pallet in order to be handled by any device in the FMS. The fixing of raw parts on pallets is executed in the load/unload station which is modelled as any other station of the

system. The selection of which parts to load on pallets is managed by the Control unit of the load/unload Station using a rule in the Dispatching rule class. When the Pallet has been assembled, it is sent to the output Secondary buffer of the loading Station and a transport request is sent to the Supervisor. The unloading task is the dual of the loading: finished parts are removed from the pallet and can leave the FMS.

The task of transporting a pallet from a station to another is managed by the Supervisor, which receives the requests, decides which request has to be satisfied first, and assigns tasks to the part handling sub-system controlled at the local level by its Control unit. The path of pallets (i.e. the sequence of stations each pallet has to visit) is decided at the system level by the Supervisor on the basis of the Routing rule.

A necessary condition for the starting of the Pallet processing is that the pallet is available in the waiting position modelled by the Secondary buffer of a Station. Once the Machine of that Station is idle, the Control unit of the Station selects a Pallet among those that are waiting in the Secondary buffer. The rule selection is taken from the Dispatching rule class. When the selected Pallet is in the working position and the necessary Tools are available to the Machine, the processing of the Pallet can start; now, the Pallet is under working and the Machine is no more idle. When the Machine finishes all the operations, assigned by the Supervisor, on the Pallet, the Supervisor selects the next destination of the Pallet by applying a Routing rule and allows the Machine to unload the pallet into the output Secondary buffer of the Station. Then, a pallet transport request is sent to the Supervisor.

4 From UML to TRIO

The UML model, partially described in the previous section, gives us a semi-formal representation of a FMS. However, if we want to carry out a quantitative analysis on the system performance it is necessary to translate the UML model in a formal model. The common practice in manufacturing is to build a simulation model to study numerically the system; instead, in this section we show how to use the TRIO language as a modeling tool, different than simulation, to analyze the main FMS properties. Notice that the shown formal approach is complementary, rather than alternative, to simulation: it does permit a deeper analysis of certain aspects of the system, and gives *certain* (i.e. formally proved), rather than *probable*, results.

TRIO [2] is a general-purpose specification language that is suitable for modeling systems with significant temporal constraints; it is a temporal logic that supports a metric notion of time. In addition to the usual propositional operators and quantifiers, it has a single basic modal operator, called $Dist$, that relates the *current time*, which is left implicit in the formula, to another time instant: given a formula F and a term t indicating a time distance, the formula $Dist(F, t)$ specifies that F holds at a time instant at t time units (with $t \gtreqless 0$) from the current instant. A number of *derived temporal operators* can be defined from the basic $Dist$ operator through propositional composition and first order quantification on variables representing a time distance. For example operators $Lasted$, $WithinP$, Alw and $Since$ are defined, respectively, as $Lasted(F,t) \triangleq \forall d\ (0 < d < t \Rightarrow Dist(F,-d))$, $WithinP(F,t) \triangleq \exists d\ (0 < d < t \wedge Dist(F,-d))$, $Alw(F) \triangleq \forall d\ (Dist(F,d))$, and $Since(F,G) \triangleq \exists d$

$(d > 0 \wedge Lasted(F, d) \wedge Dist(G, -d))$. TRIO is well-suited to deal with both continuous and discrete time, with only minor changes in the definition of some operators. In addition, TRIO has the usual object-oriented concepts and constructs such as classes, inheritance and genericity, which are suitable for specifying large and complex systems.

Figure 2 shows the graphical representation of a subset of the TRIO specification that corresponds to the FMS semi-formal model presented in Section 3. It depicts the TRIO classes representing, respectively, the supervisor (box SV), the transport network (box TN), and the set of stations that perform operations on the raw parts (boxes ST). Figure 2 shows that the supervisor class SV can send the stations (array ST) a process_pallet event, and that it contains an internal (i.e. not visible outside the class) state processing[3].

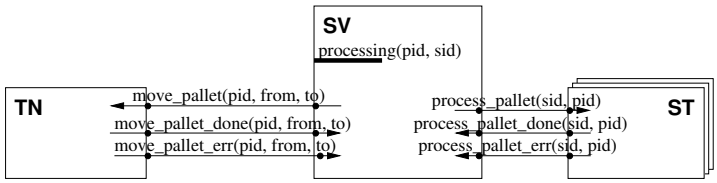

Fig. 2. TRIO classes corresponding to a subset of the FMS model

The behavior of the components of the system is defined by the *axioms* belonging to the corresponding TRIO classes. In our example, let us assume that the pallets are identified by a positive integer in the range $\mathrm{PID} = [1..N_p]$, while stations are identified by a positive integer in the range $\mathrm{SID} = [1..N_s]$ (which is the domain of the indexes of array ST of Figure 2). The following axiom (where $pid \in \mathrm{PID}$ and $src, dst \in \mathrm{SID} \cup \{0\}$, with 0 representing the central part buffer) of class SV states that, if a pallet completes a move (i.e. event move_pallet_done occurs), or if there is a move error (move_pallet_err), a move_pallet command was issued in the past Δ time units.

Axiom 1 $\mathrm{move\_pallet\_done}(pid, src, dst) \vee$
$\mathrm{move\_pallet\_err}(pid, src, dst) \Rightarrow WithinP(\mathrm{move\_pallet}(pid, src, dst), \Delta)$[4].

Furthermore, let us assume that the pallets are partitioned into three subgroups A, B, C and the stations are partitioned into two subgroups D, E. Then if $pid \in \mathrm{PID}$, $isA(pid)$ (respectively $isB(pid)$, $isC(pid)$) is true if and only if pid is the identifier of a pallet belonging to subgroup A (resp. B, C). In a similar way, if $sid \in \mathrm{SID}$, $isD(sid)$ (resp. $isE(sid)$) is true if and only if sid is the identifier of a station belonging to subgroup D (resp. E). Axiom 2 of class SV states that when a pallet is moved from a source to a station, some compatibility constraints are enforced; more precisely, pallets of type A can only be moved to machines of subgroup D, while pallets of type C can only be moved to stations of subgroup E (pallets of type B, instead, can be moved anywhere).

[3] Informally speaking, an event is a predicate that holds only in isolated time instants, while a state is a predicate that holds in intervals of non-null duration (see [3]).

[4] TRIO formulae are implicitly temporally closed with the Alw operator.

Axiom 2 $\text{move\_pallet}(pid, src, dst) \Rightarrow$
$(isA(pid) \Rightarrow isD(dst)) \wedge (isC(pid) \Rightarrow isE(dst))$

The structure of the TRIO description (i.e. the division in classes and corresponding items, events, and states) can be derived from the UML class diagram through (rather loose) methodological considerations. TRIO axioms (that is, the behavior contained in the classes) can then be written by taking into consideration, among other things, UML behavioral diagrams (sequence diagrams, etc.). Obviously, UML diagrams do not contain all the required information to build a completely formal model; therefore, the translation process cannot be completely automated. Nonetheless, suitable tools can manage most low-level details, thus restricting the amount of required interaction to a minimum (namely comparable to that required in building a simulation model).

The basic idea when passing from UML to TRIO is to transform UML classes into TRIO classes. Then, from the UML diagram of Figure 1 we determine that there must be a TRIO class to model the supervisor and a TRIO class to model the machining centers (other classes could be introduced, but are not included in the current description for the sake of brevity and simplicity).

The methods of the UML classes, instead, are modeled in TRIO through combinations of states and events. For example, UML class Transport Network has a method move_pallet (not shown here for the sake of brevity), which is translated into events move_pallet and move_pallet_done, representing, respectively, the invocation and successful response of the method. Event move_pallet_err, instead, represents an incorrect termination of (i.e. an exception being raised by) method move_pallet. The basic semantics of UML methods is then captured by TRIO axioms on the corresponding events. For example, Axiom 1 shown above describes a most natural property of method move_pallet and, more precisely, that a method terminates (successfully or not) only if it was previously invoked.

In addition, the TRIO counterpart of a UML method might also include a state that models when a method is executing. For example, state processing of class SV is true when method process_pallet is computing. This is represented by the following axiom of class SV, which defines that state processing is true only if neither event process_pallet_done nor event process_pallet_err has occurred since event process_pallet occurred.

Axiom 3 $\text{processing}(pid, sid) \Rightarrow Since(\neg(\text{process\_pallet\_done}(sid, pid) \vee$
$\text{process\_pallet\_err}(sid, pid)), \text{process\_pallet}(sid, pid))$

Other axioms are not related to the basic behavior of single methods, but, rather, to how classes combine methods to achieve the desired results. This kind of information can be extracted from UML sequence diagrams and statecharts, following certain methodological guidelines. For example, a sequence diagram, not shown here for the sake of brevity, says that a pallet starts being processed when it completes a successful move into a station. This is modeled by the following axiom of class SV:

Axiom 4 $\text{process\_pallet}(sid, pid) \Rightarrow \exists src \in SID : \text{move\_pallet\_done}(pid, src, sid)$

Armed with the formal model presented above, it is possible to formally prove properties of the modeled system, as the next section will show.

5 Verification of the Formal Model

Let us consider a very simple property verification for the TRIO model of the FMS system. Fragments of the formal specification of the model have been introduced in Sections 3 and 4. From those axioms, we are able to prove a simple property by deducing its validity via theorem-proving. Obviously, other approaches to formal verification could be chosen (e.g. model-checking). Moreover, it is likely that in the overall verification of a complex description of a FMS different techniques are combined to compose local properties into a complete global verification. Notice that this kind of analysis is not possible with a simulation model (which only gives statistical results) or with the UML model alone (which lacks a non-ambiguous semantics).

The property we consider in this simple example is stated informally as following: whenever a pallet pid is being processed in a station sid, the compatibility constraint that every pallet of type A (resp. C) is processed in stations of type D (resp. E) is respected for the pair pid, sid. This means that the constraint enforced on the moving of pallets (as in Axiom 2) is also guaranteed during the processing of pallets.

The following theorem formally states the above (partial) correctness requirement.

Theorem 5 (partial_correctness) $\mathrm{processing}(pid, sid) \Rightarrow$
$(isA(pid) \Rightarrow isD(sid)) \wedge (isC(pid) \Rightarrow isE(sid))$

Let us sketch the proof of theorem partial_correctness. Let us consider a generic $pid \in PID$ such that $isA(pid)$. Assuming $\mathrm{processing}(sid, pid)$ holds at the current time t, we have to show that $isD(sid)$. By considering Axiom 3, we deduce that $Since(\neg(\mathrm{process\_pallet\_done}(sid, pid) \vee \mathrm{process\_pallet\_err}(sid, pid)), \mathrm{process\_pallet}(sid, pid))$ holds at t. If we consider the definition of the $Since$ operator, this means that $\exists u < t$ such that $Lasted(\neg(\mathrm{process\_pallet\_done}(sid, pid) \vee \mathrm{process\_pallet\_err}(sid, pid)), t - u)$ holds at t and $\mathrm{process\_pallet}(sid, pid)$ holds at u. Let us focus on the latter term: $\mathrm{process\_pallet}(sid, pid)$ is true at u. Therefore, we can use Axiom 4 to infer that $\mathrm{move\_pallet\_done}(pid, src, sid)$ is also true at u, for some $src \in SID$. This also makes true the antecedent of the implication in Axiom 1, so that we deduce that $WithinP(\mathrm{move\_pallet}(pid, src, sid), \Delta)$ holds at u. By the definition of the $WithinP$ operator, we immediately infer that there exists an instant $v \in (u - \Delta, u)$ such that $\mathrm{move\_pallet}(pid, src, sid)$ holds at v. Now, we consider Axiom 2, which relates $\mathrm{move\_pallet}$ events to the type of its arguments. We conclude that $isD(sid)$, since $isA(pid)$. This means that the theorem holds for all $pid \in A \subseteq PID$. The cases for $pid \in B$ and for $pid \in C$ are very similar and are omitted. □

6 Conclusions and Future Works

In this paper we applied modeling techniques typical of the software development domain to flexible manufacturing systems. Our goal was twofold: on the one hand we developed a UML-based description of FMSs, which is an intermediate representation between informal specifications written in natural language and simulation models written in some object-oriented simulation language; on the other hand, from this UML description we derived a formal model expressed in the TRIO temporal logic, which was used to formally analyze a part of the system with deductive verification techniques.

This paper is a just a preliminary result of a large project, and a variety of future developments will follow in the next future. First, a complete methodology to derive simulation models from UML and TRIO descriptions would greatly enhance the design of a FMS. This would both simplify the development process of simulation models, and provide users with two complementary descriptions, which would allow a more comprehensive and thorough analysis of the properties of the analyzed FMS. Armed with this methodology, the user could map TRIO concepts to simulation elements (and vice versa), thus improving the interactions between the two models.

A natural extension of our approach would be to prove more complex properties of the system and to validate the implemented FMS using the TRIO model. This can be done in different ways. For example, test cases (that is, execution traces) could be (semi)automatically derived from the TRIO description [6], and then compared with the behavior of the actual FMS. Vice versa, one could capture both the most common and least common behaviors of the implemented system and check if they are compatible with the TRIO model (in TRIO terms, this is called *history checking*).

While in this paper we focused on simple properties of single parts of a FMS, TRIO is a flexible formalism that can be used to verify a number of different (local and global) types of properties. For example, one could formally compare different management policies (scheduling, loading of parts, tool dispatching, etc.) to determine which ones should be adopted in a specific FMS. Another interesting application would be to verify, with respect to a certain overall goal, the consistency of the several management rules introduced in the simulation model by the developer.

Finally, in a sort of "automatic design" approach [2], we plan on using the TRIO model to prove that, for some property to hold (e.g. the pallet flowtime being lower than a specified value, etc.), FMS configuration parameters must be in some precise range.

Acknowledgements

The authors would like to thank Dino Mandrioli, Pierluigi San Pietro, Quirico Semeraro and Tullio Tolio for their useful suggestions and comments, and the anonymous reviewers for their suggestions for improvements.

References

1. A. Matta. D1.1: UML description of FMS. Technical report, FIRB Project, Contract code: RBNE013SWE, 2004.
2. E. Ciapessoni, A. Coen-Porisini, E. Crivelli, D. Mandrioli, P. Mirandola, and A. Morzenti. From formal models to formally-based methods: an industrial experience. *ACM TOSEM*, 8(1):79–113, 1999.
3. Angelo Gargantini and Angelo Morzenti. Automated deductive requirement analysis of critical systems. *ACM TOSEM*, 10(3):255–307, July 2001.
4. A. Matta, T. Tolio, F. Karaesmen, and Y. Dallery. An integrated approach for the configuration of automated manufacturing systems. *Robotics and CIM*, 17(1-2):19–26, 2001.
5. OMG. UML Specification, 2003-03-01. Technical report, OMG, March 2003.
6. P. San Pietro, A. Morzenti, and S. Morasca. Generation of execution sequences for modular time critical systems. *IEEE Transactions on Software Engineering*, 26(2):128–149, 2000.