Analyzing Timing Constraints in Flexible Manufacturing Systems

Stephan Flake*, Wolfgang Mueller*, Ulrich Pape**, Juergen Ruf***

* C-LAB, Paderborn University, Fuerstenallee 11, 33102 Paderborn, Germany
** Heinz Nixdorf Institut, Paderborn University, Fuerstenallee 11, 33102 Paderborn, Germany
*** Wilhelm-Schickard Institut, Tuebingen University, Sand 13, 72076 Tuebingen, Germany

Abstract

This article investigates the formal verification of MFERT models, an industry-approved graphical notation and methodology for the specification of manufacturing systems. We demonstrate our approach by the example of an automatic transport system within the context of a flexible manufacturing plant. For formal verification we apply the RAVEN model checker. RAVEN checks time-critical properties on networks of time-annotated Finite State Machines. Time-critical properties are specified by Clocked CTL (CCTL) formulae. Additional timing analysis enables users to gain detailed information on quantitative timing properties.

1 Introduction

In recent years model checking has been successfully applied to verify the correctness of hardware and software designs, in particular in the field of digital circuits and communication protocols. The most remarkable advantage of model checking is that the task of verifying a model for certain properties is fully automated. Moreover, a model checker typically generates counter examples in cases when the model does not satisfy a specified property. A counter example demonstrates an execution of the model that leads to a situation which falsifies the property, which is very helpful for detailed error analysis. In time-dependent models, such as for manufacturing systems, the correct time-critical behavior of certain properties is of particular interest. Nevertheless, only few tools support the formal verification of such models. Instead, simulation is still heavily applied to validate the correct behavior of models for manufacturing systems.

In this paper, we present an approach which covers the design flow for the modelling and verification of manufacturing systems. In a first step, the designer specifies a model in a graphical specification language (MFERT). The MFERT description, i.e., the model, is translated into an annotated Finite State Machine based formalism (I/O-interval structures) for model checking. For this model, properties are specified in Clocked CTL for formal verification with the RAVEN model checker [15]. For advanced timing analysis, RAVEN provides additional facilities to check for min, max, and stable times.

This paper is organized as follows. The next section discusses related works in the domain of model checking focusing on manufacturing systems. Section 3 gives an overview of MFERT. In Section 4, we sketch the formal RAVEN model of I/O-interval structures focusing on the introduction of the temporal logic CCTL. Section 5 introduces code generation of I/O-interval structures from MFERT. In Section 6, we identify meaningful properties for manufacturing system verification and give their representation in Clocked CTL and query formulae, respectively. Section 7 presents some preliminary results, before Section 8 concludes with remarks and an outlook to future work.

2 Related Works

Formal verification of manufacturing systems with a focus on the analysis of real-time properties has not been widely adopted by industry yet. Present practice in general systems design is still that mainly simulation is applied to detect qualitative and safety-critical defects in system specifications, e.g., [10][11][19].

Formal verification in industrial context is currently mainly applied for rather small systems and subsystems. In general, verification tools are mainly available for discrete systems analysis, e.g., VERUS [3]. Only few are based on a continuous time model, e.g., HyTech for hybrid automata [7]. They investigate static timing analysis on larger models of restricted (strongly linear) hybrid systems. That work, for instance, is limited to state space exploration and does not cover verification of properties.

Methods for the formal verification of discrete systems are barely applied to model manufacturing systems right now. There are verification approaches in the domain of chemical engineering. But they are not dedicated to the analysis of timing properties [17]. Very few approaches exist on the formal verification of real-time properties in the field of electronic systems [1].

Quite a couple of verification tools have already been integrated into design frameworks for their application in specific domains. PEP [6], for instance, is a Petri-Net based tool that provides a graphical user interface for the design of parallel systems. The models have a well-defined internal format so that they can be easily adopted to different model checkers, like SPIN and SMV. In another approach, the model checker SPIN is used to analyze AMBER models [9]. AMBER is a graphical notation dedicated to the domain of business processes. A formal semantics is defined by a mapping to PROMELA, the input language of SPIN. Again, this approach does not consider timing analysis. Other environments support MSC's, StateCharts, and SDL as a graphical front-end for formal analysis [2].

3 MFERT

We are using the structure-oriented language MFERT as the basis for modelling manufacturing processes. MFERT is short for "**M**odell der **FERT**igung" (German for: Model of Manufacturing). MFERT provides means and a methodology for specification and implementation of planning and control assignments in manufacturing processes. MFERT is a universal approach which has been successfully applied in different projects with various industrial partners [8], additionally acknowledged by the German science award of logistics [16].

An MFERT model is based on production elements (F-Elements) and production processes (F-Processes). Production elements represent objects, whose properties are changed by processes and transformations. Properties of production elements are described by attributes. A production elements obtains its own identity, composed out of the description and the element's correlative status. Using this identity, the different states during the production can be associated to production elements.

An MFERT model is a directed bipartite graph of FE-nodes and FP-nodes. The graphical notation of an FE-node is a triangle △, and FP-nodes are given by rectangles □. Each FE-Node represents a specific state and can be seen as a container for F-Elements in the respective state. FP-nodes represent transformations on F-Elements, performed by F-processes. Nodes are connected by edges that describe exchange relations between two nodes. Edge annotations define if a predecessor is bringing, providing, or waiting for elements and processes, as well as when a successor is fetching, receiving, or waiting for elements. Interface edges are for connecting different levels of hierarchy. They additionally allow the coupling to a real production environment.

Figure 1 gives an MFERT example with table-tops, table-legs, and screws, which are combined to complete tables.

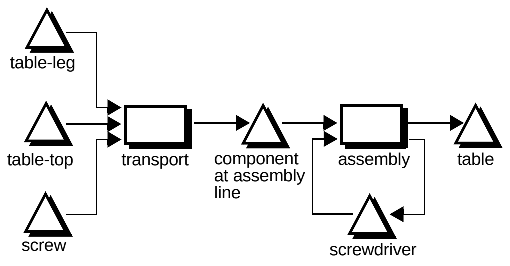

Figure 1. Example MFERT Diagram for a Table Production Assignment

F-Elements and F-Processes are in certain states, which are characterized by attributes. An attribute denotes a feature of a model element and assigns a value to a relation. Constructors for the definition of discrete time modes are available. In practice, different time models are often necessary for the definition of production assignments, e.g., the provision of the source materials for an assembly line may take place non-recurringly at the beginning of a shift, while the mounted end products are transported every hour to the delivery store.

MFERT is dedicated to the specification and implementation of automatic production systems in full complexity. For implementation, model nodes are equipped with functions, and their process control is carried out by means of message exchange between nodes and by a so-called "global manager" that coordinates the computations in the model.

4 Real-Time Model Checking

For formal verification of MFERT models, we apply the RAVEN model checker. In RAVEN, the model is given by a time-annotated state transition system, i.e., a set of I/O-interval structures. Interval structures are based on Kripke structures with [min,max]-time intervals at their state transitions. This section briefly sketches the basics of interval structures before outlining CCTL. For a more detailed introduction to I/O-interval structures the reader is referred to [13].

An interval structure $\Im$ is a tuple (P, S, T, L, I) with a set of propositions P, a set of states S, a transition relation T between the states, such that every state in S has a successor state, a state labeling function $L: S \to \wp(P)$, and a transition labeling function $I: T \to \wp(\mathbb{N})$. I/O-interval structures are interval structures with I/O-operations. We assume that each interval structure has exactly one clock for measuring time. The clock is reset to zero, if a new state is entered. A state may be left, if the actual clock value corresponds to a delay time labelled at an outgoing transition. The state must be left, if the maximal delay time of all outgoing transitions is reached. A configuration of an interval structure $g = (s, v)$ is a state s associated with a clock value v. The set of all valid configurations in $\Im$ is called G.

Clocked Computational Tree Logic (CCTL) is for the specification of properties for a given interval structure [12]. CCTL formulae are composed from propositions denoting predicates in combination with Boolean connectives and time-annotated temporal operators. The CCTL syntax is as follows:

$$\varphi := \begin{array}{l} p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \\ \mid \mathsf{EX}_{[a]}\varphi \mid \mathsf{EF}_{[a,b]}\varphi \mid \mathsf{EG}_{[a,b]}\varphi \\ \mid \mathsf{E}(\varphi\ \mathsf{U}_{[a,b]}\varphi) \mid \mathsf{E}(\varphi\ \mathsf{C}_{[a]}\varphi) \mid \mathsf{E}(\varphi\ \mathsf{S}_{[a]}\varphi) \\ \mid \mathsf{AX}_{[a]}\varphi \mid \mathsf{AF}_{[a,b]}\varphi \mid \mathsf{AG}_{[a,b]}\varphi \\ \mid \mathsf{A}(\varphi\ \mathsf{U}_{[a,b]}\varphi) \mid \mathsf{A}(\varphi\ \mathsf{C}_{[a]}\varphi) \mid \mathsf{A}(\varphi\ \mathsf{S}_{[a]}\varphi) \end{array}$$

where $p \in P$ is an atomic proposition and $a \in I\!N_0$ and $b \in I\!N_0 \cup \{\infty\}$ are time-bounds. All interval operators can also have single time-bound only. If not specified, the lower bound is zero and the upper bound is infinite by default. In the case an X-operator has no time bound specification, its default value is one.

The semantics of CCTL is defined as a validation relation "$\models$" as follows, using the notion of *runs*, which are possible sequences of configurations that occur during execution of $\Im$.

Definition 4.1. *Given an interval structure $\Im$ and a configuration $g_0 = (s, v) \in G$.*

$$\begin{array}{lll} g_0 \models p & :\Leftrightarrow & p \in L(s) \\ g_0 \models \neg\varphi & :\Leftrightarrow & \text{not } g_0 \models \varphi \\ g_0 \models (\varphi \wedge \psi) & :\Leftrightarrow & g_0 \models \varphi \text{ and } g_0 \models \psi \\ g_0 \models \mathsf{EG}_{[a,b]}\varphi & :\Leftrightarrow & \text{there ex. a run } r = (g_0, \ldots) \text{ s.t.} \\ & & \text{for all } a \le i \le b \text{ holds } g_i \models \varphi \\ g_0 \models \mathsf{E}(\varphi\ \mathsf{U}_{[a,b]}\psi) & :\Leftrightarrow & \text{there ex. a run } r = (g_0, \ldots) \\ & & \text{and an } a \le i \le b \text{ s.t. } g_i \models \psi \\ & & \text{and for all } j < i \text{ holds } g_j \models \varphi \\ g_0 \models \mathsf{E}(\varphi\ \mathsf{C}_{[a]}\psi) & :\Leftrightarrow & \text{there ex. a run } r = (g_0, \ldots) \text{ s.t.} \\ & & \text{if for all } i < a \text{ holds } g_i \models \varphi \\ & & \text{then } g_a \models \psi \\ g_0 \models \mathsf{E}(\varphi\ \mathsf{S}_{[a]}\psi) & :\Leftrightarrow & \text{there ex. a run } r = (g_0, \ldots) \text{ s.t.} \\ & & \text{f.a. } i < a \text{ holds } g_i \models \varphi \text{ and } g_a \models \psi \end{array}$$

Other operators can be derived by the previous ones, such as:

$$\begin{array}{lcl} \mathsf{EF}_{[a,b]}\varphi & \equiv & \mathsf{E}(true\ \mathsf{U}_{[a,b]}\varphi) \\ A(\varphi\ \mathsf{C}_{[a]}\psi) & \equiv & \neg\mathsf{E}(\varphi\ \mathsf{S}_{[a]}\neg\psi) \\ A(\varphi\ \mathsf{S}_{[a]}\psi) & \equiv & \neg\mathsf{E}(\varphi\ \mathsf{C}_{[a]}\neg\psi) \end{array}$$

Additional timing analysis queries help users to extract important time bounds from formal system descriptions. For instance, one might be interested in the maximal number of time steps a workpiece is waiting until it is processed. Other typical problems are minimal and maximal delay times between events, e.g., the maximal time until the first workpiece leaves the process. Three algorithms, which are suitable to investigate such quantitative timing analysis questions, are formally captured by the following definition.

Definition 4.2. *Given an interval structure $\Im$ and a set of configurations $R \subseteq G$ [1] (the "region" set). The function $STABLE$: $\wp(G) \rightarrow I\!N$ is defined through:*

$$\mathsf{STABLE}(R) := \begin{cases} m & \text{if } m = max\left\{ \begin{array}{c} n \mid \exists g \in R, \exists r \in \Pi(g). \\ \forall i < n\,.\, r_i \in R \end{array} \right\} \\ 0 & \text{if } R = \emptyset \\ \infty & \text{otherwise} \end{cases}$$

The function MIN : $\wp(G) \times \wp(G) \rightarrow I\!N$ is defined through:

$$\mathsf{MIN}(F, D) := \begin{cases} m & \text{if } m = min\left\{ \begin{array}{c} n \mid \exists g \in F, \exists r \in \Pi(g). \\ r[n] \in D \end{array} \right\} \\ \infty & \text{otherwise} \end{cases}$$

The function MAX : $\wp(G) \times \wp(G) \rightarrow I\!N$ is defined through:

$$\mathsf{MAX}(F, D) := \begin{cases} m & \text{if } m = max\left\{ \begin{array}{c} n \mid \exists g \in F, \exists r \in \Pi(g). \\ r[n] \in D \wedge \\ \forall i < n\,.\, r[i] \notin D \end{array} \right\} \\ \infty & \text{otherwise} \end{cases}$$

In the context of RAVEN, an interval structure and a set of CCTL formulae have to be specified by the means of the RAVEN Input Language (RIL). A RIL specification contains a set of global definitions (e.g., fixed time bounds or frequently used formulae) followed by the specification of parallel running modules and CCTL specifications plus timing analysis queries.

1. We assume that G contains only reachable configurations.

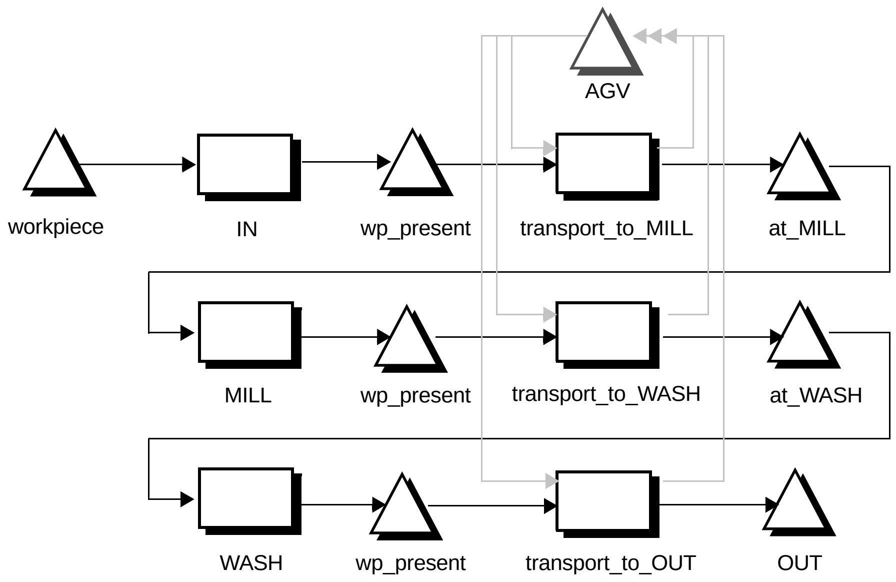

Figure 2. MFERT Diagram of Manufacturing Case Study

5 Model Checking of MFERT

For applying RAVEN to MFERT models we have to introduce a transformation of MFERT to I/O-interval structures. We illustrate this transformation by a case study taken from the domain of flexible manufacturing systems.

The production process is defined by the MFERT diagram in Figure 2: Raw workpieces (wp*s*) enter the manufacturing plant at an input storage IN, are transported from IN to machine MILL by an automated guided vehicle (AGV), then further carried on to machine WASH (possibly by a different AGV), and finally approaches at output storage OUT.

Subprocesses can be modelled by refined MFERT diagrams, e.g., the process transport_to_MILL can be divided into (a) loading an AGV at IN, (b) AGV moving the workpiece to machine MILL, (c) unloading the AGV at MILL. In the following, we only investigate step (a) and assume that there are always workpieces available for loading, according to the case study description in [18]. A cyclic process at station IN (1) loads an AGV with a workpiece and (2) provides the next workpiece for loading (cf. Figure 3).

For the subprocess of loading at station IN an initial task has to be specified. This is marked by a double framed triangle. In our example wp_present is initial, indicating that a workpiece is ready to be loaded. For the process of loading, an AGV is delayed by time annotations which are assigned to the outgoing edges. Note here, that AGV is taken from the top-level MFERT diagram, while node AGV_loaded_at_IN is added to the model as a new element.

Generating I/O-Interval Structures. Given the previously introduced MFERT models, we now demonstrate their translation to interval structures in RIL (RAVEN Input Language).

For each MFERT subdiagram a separate I/O-interval structure is required (cf. Figure 3). Each MFERT node is represented by a state. Edges are interpreted as time-annotated state transitions. Production elements that are

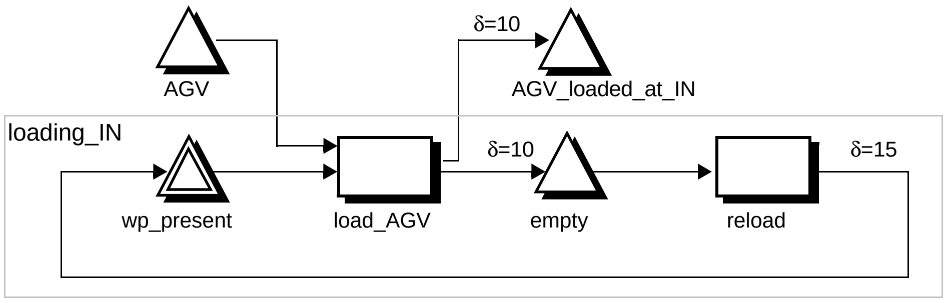

Figure 3. MFERT Diagram for Loading at Station IN

```
MODULE load_at_IN
          SIGNAL s : { wp_present, load_AGV, empty, reload }
          INPUTS AGV := h1.at_IN || h2.at_IN || h3.at_IN
          DEFINE AGV_loaded_at_IN := (s==empty)
          INIT      wp_present
          TRANS |- s == wp_present  -- AGV                : 1  -->  s := load_AGV
                |- s == wp_present  -- !AGV               : 1  -->  s := wp_present
                |- s == load_AGV    --                    : 10-->  s := empty
                |- s == empty       --                    : 1  -->  s := reload
                |- s == reload      --                    : 15-->  s := wp_present
END
```

Figure 4. RIL-Code for Loading at Station IN

connected by incoming edges to the MFERT subdiagram (e.g., AGV in Figure 3) are interpreted as *conditions*. In our example, the condition is formulated as a Boolean expression w.r.t. 3 available AGVs:

```
AGV := h1.at_IN || h2.at_IN || h3.at_IN
```

Conditions must be completely specified when edges are mapped to state transitions. For example, when considering the condition "in state wp_present and AGV is true" the alternative condition "in state wp_present and AGV is false" must be added to get a valid description of an I/O-interval structure.

The nodes of the MFERT subdiagram are basically mapped to binary coded state variables. The corresponding RIL module of the example is given in Figure 4. The internal signals, inputs, and local macros (DEFINE) are given in the first part. They are followed by the definition of the initial states and the state transitions. The latter have the form:

```
|- state -- condition : delay --> next_state
```

Figure 4 gives only an excerpt from the complete specification. The RIL code for AGVs with more than 50 states for maintaining the navigation can be found in [14]. The complete generation requires some more composition and reduction algorithms to retrieve a single interval structure out of the modules. Details about composition and model checking algorithms are available in [13].

6 Property and Timing Analysis

The generation of RIL from MFERT enables the automatic formal verification through the RAVEN model checker. We now outline the specification of properties that are of specific interest in the context of our manufacturing case study. We present some meaningful definitions and give the corresponding CCTL formulae. The presented formulae can be easily rephrased for other situations in the manufacturing process, i.e., for different AGVs, their positions, other machines, and different timing values.

We start with a simple property about the location of all AGVs. A heading AG-operator in the formula defines that the property should hold on **A**ll possible execution paths **G**lobally at all time steps, i.e., the property should be valid all the time during system execution.

Property 1: There is at all times at most one AGV at station IN .

$$\begin{aligned}&\mathrm{AG}\;(\\&(\mathrm{h1.at\_IN} \rightarrow \neg(\mathrm{h2.at\_IN} \vee \mathrm{h3.at\_IN}))\wedge\\&(\mathrm{h2.at\_IN} \rightarrow \neg(\mathrm{h1.at\_IN} \vee \mathrm{h3.at\_IN}))\wedge\\&(\mathrm{h3.at\_IN} \rightarrow \neg(\mathrm{h1.at\_IN} \vee \mathrm{h2.at\_IN})))\end{aligned}$$

This formula defines the mutual exclusion for AGVs at input IN: If one of the AGVs is currently at IN, then the other two must not show up there. Correspondingly, other positions at machines and crossings along transportation paths are checked. All these properties together ensure that the AGVs will never collide.

Property 2: Each workpiece must be picked up at input IN not later than 400 time units after it has been provided.

$$\begin{aligned}&\mathrm{AG}\;(\mathrm{loading\_IN.wp\_present} \rightarrow\\&\qquad \mathrm{AF}_{[400]}(\neg\mathrm{loading\_IN.wp\_present}))\end{aligned}$$

If there is a workpiece available to be picked up, then on all possible further execution paths (**A** of AF-operator) it must happen at some time (**F** of AF-operator) in future within the next 400 time units that the given state is left.

Property 3: Machine MILL must not be idle for more than 400 time units.

$$\mathrm{AG}\;(\neg\mathrm{MILL.processing} \rightarrow \mathrm{AF}_{[400]}(\mathrm{MILL.processing}))$$

This formula is similar to the one of property 2, except that the negation is reversed.

Property 4: The interval between completed workpieces being unloaded at the output OUT is at most 500 time units.

$$\mathrm{AG}\;(\mathrm{OUT.full} \wedge \mathrm{AX}(\neg\mathrm{OUT.full} \rightarrow \mathrm{AF}_{[1,\,500]}(\mathrm{OUT.full})))$$

The first part of this formula expresses the condition that must hold: We are checking the time point when a workpiece has been unloaded to OUT (OUT.full) and in the next step (AX-operator with implicit timing interval 1) OUT is ready to receive new input again, i.e., the buffer is not full. If this is true, the buffer must be filled again before the next 500 time units (AF-operator with time interval [1,500]).

Checking these properties with the RAVEN model checker, the result is either "yes" (property is valid) or "no" (property is not valid). In the latter case, a counter example trace of the model execution is given and can be viewed in a wave form browser[1]. This supports the designer to detect and correct errors in the model or to rephrase the property specification. In order to support this iterative process of verification, additional timing analysis queries can be formulated. Answers to such queries give information about times required for execution of specific tasks. The following examples give meaningful timing analysis specifications in the context of our case study.

Timing Analysis 1: How many time units are required to receive the first workpiece at the output storage OUT?

MIN (INIT, OUT.wp_present)

Here, INIT represents the initial system configuration. The target system configuration is specified in the second parameter: OUT.wp_present.

Timing Analysis 2: When does AGV h1 unload a workpiece at the output OUT for the first time?

MIN (INIT, h1_at_OUT & OUT.wp_present)

This formula is similar to the first analysis query above. Here, we investigate a specific AGV unloading at station OUT. h1_at_OUT represents a Boolean formula that summarizes the states of AGV h1 when it reaches station OUT. This has to be combined with the condition OUT.wp_present to ensure that h1 is really unloading a workpiece and is not just passing by.

Timing Analysis 3: What is the minimum time interval between workpieces that are unloaded at the output storage OUT?

$$\text{MIN (OUT.wp\_present} \wedge \text{EX} \neg\text{OUT.wp\_present,} \\ \neg\text{OUT.wp\_present} \wedge \text{EX OUT.wp\_present)}$$

We are starting from a configuration in which a workpiece has been loaded into the input buffer of OUT and is now removed from there, using the EX-operator to declare two subsequent configurations. Similarly, the second parameter specifies the configuration, in which a workpiece is going to be loaded into the input buffer.

1. We are currently developing a translation of these traces to instructions for an existing 3D animation of the case study in order to provide an intuitive visualization of such counter examples.

Timing Analysis 4: What is the maximum unloading interval of workpieces at the output storage OUT?

$$\text{MAX (OUT.wp\_present} \wedge \text{EX} \neg\text{OUT.wp\_present,} \\ \neg\text{OUT.wp\_present} \wedge \text{EX OUT.wp\_present)}$$

Correspondingly, the maximum time between two system configurations can be computed.

Timing Analysis 5: How long must a workpiece be waiting at the most to be picked up at station IN?

STABLE(IN.wp_present)

This query can be easily formulated through the STABLE function. Similarly, all other stations and machines can be checked.

Timing Analysis 6: How long is machine MILL idle at most?

STABLE(!MILL.processing)

This query defines by negation all states of machine MILL in which it is not working on a workpiece, i.e., all states that are different from processing.

Although the properties and queries presented in this paper are relatively easy to specify in temporal logics formulae or analysis algorithm calls, the formulation of properties in general is a task too cumbersome for people not trained in formal methods. Based on research on recurring patterns in existing property specifications [4], we have developed a set of useful structured English sentences to formulate typical time-critical properties and timing analysis queries. To avoid ambiguities, the sentences are given a formal semantics by their translation to CCTL formulae [5].

7 Results

So far, we have manually transformed a complete MFERT model of the case study and analyzed the resulting I/O-interval structures with RAVEN. We could verify that the AGVs indeed do not collide in our MFERT model[2]. Additional queries have proven that the first workpiece is finished after 890 time units, and that afterwards the interval between two completed workpieces unloaded at the output storage is between 209 and 229 time units. The longest time for a machine to be idle is 136. More details can be found in [14].

On a PC with a 366MHz processor and 96MB RAM under Linux, the automatic verification of the model with RAVEN took less than one minute for analyzing possible collisions and some timing queries.

2. Under the assumption that AGVs are moving along paths defined by fixed positions.

8 Conclusion

The translation from MFERT to model checking input presented in this article is a first step for applying automated verification of models with property specification and timing analysis to MFERT models. Not all of the MFERT concepts could be integrated into the translation so far and we are currently limited to a synchronous interpretation of MFERT. Nevertheless, the presented studies showed that the RAVEN model checker can be used to verify qualitative properties of modular MFERT models including timing issues. We are expecting an even better applicability of the upcoming next version of RAVEN since it includes the analysis of quantitative properties, e.g., for the dimensioning of input and output buffers.

The presented results are based on a manual transformation of an MFERT diagram modelling a flexible manufacturing system. Since the works on our graphical MFERT capture are almost finished, RIL code generation will be integrated next.

Acknowledgements

This work has been supported through the DFG project GRASP within the DFG Schwerpunktprogramm 1064 "Integration von Techniken der Softwarespezifikation fuer ingenieurwissenschaftliche Anwendungen".

Literature

[1] M. Bozga, O. Maler, A. Pnueli, S. Yovine: Some Progress in the Symbolic Verification of Timed Automata. In O. Grumberg, ed., *Computer Aided Verification*, pp. 179-190, LNCS 1254, 1997, Springer-Verlag.

[2] D. Bosnacki, S. Mauw, T. Willemse. *Proc. of the First International Symposium on Visual Formal Methods*. Report 99/08, Dept. of Mathematics and CS, Eindhoven University of Technology, 1999.

[3] S. V. Campos, E. M. Clarke, M. Minea. The Verus Tool: A Quantitative Approach to the Formal Verification of Real-Time Systems. In O. Grumberg, ed., *Computer Aided Verification*, pp. 452-455, LNCS 1254, 1997, Springer-Verlag.

[4] M. B. Dwyer, G. S. Avrunin, J. C. Corbett. Property Specification Patterns for Finite-State Verification. In *Proc. of the 2nd Workshop on Formal Methods in Software Practice*, pp. 7-15, Clearwater Beach, Florida, USA, 1998.

[5] S. Flake, W. Mueller, J. Ruf. Structured English for Model Checking Specification. In *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, GI/ITG/GMM Workshop*, Frankfurt/M., Germany, Feb. 2000.

[6] B. Grahlmann. The State of PEP. In A. M. Haeberer, ed., *Proc. of Algebraic Methodology and Software Technology*, LNCS 1548, Jan. 1999. Springer-Verlag.

[7] T. A. Henzinger, P.-H. Ho, H. Wong-Toi. HyTech: The Next Generation. In *Proc. of the 16th Annual IEEE Real-Time Systems Symp.*, IEEE CS Press, pp. 56-65, 1995.

[8] R. Holtkamp. Ein objektorientiertes Rahmenwerk zur Erstellung individueller, verteilter Fertigungslenkungssysteme. *Dissertation, Heinz Nixdorf Institut*, HNI-Verlagsschriftenreihe, Band 51, March 1999. (in German)

[9] W. Janssen, R. Mateescu, S. Mauw, J. Springintveld. Verifying Business Processes using SPIN. In G. Holzmann, E. Najm, A. Serhrouchni, eds., *Proc. of the 4th Int. SPIN Workshop*, pp. 21-36, Paris, France, Nov. 1998.

[10] K. Nguyen. The Development and Implementation of a Cell Controller Framework. In *Proceedings of the IEEE/SEMI Int. Semiconductor Manufacturing Science Symp.*, pp. 54-57, 1993.

[11] K. Pirklbauer, R. Plötsch, R. Weinreich. Object-Oriented Process Control Software. In *Journal of Object-Oriented Programming*, Vol. 7, No. 2, pp. 30-35, 67, SIGS Publications, New York, USA, May 1994.

[12] J. Ruf, T. Kropf. Symbolic Model Checking for a Discrete Clocked Temporal Logic with Intervals. In *Conf. on Correct Hardware Design and Verification Methods*, pp. 146–166, Montreal, Canada, Oct. 1997, Chapman & Hall.

[13] J. Ruf and T. Kropf. Modeling and Checking Networks of Communicating Real-Time Processes. In *Int. Conf. on Correct Hardware Design and Verification Methods*, pp. 265-279, Bad Herrenalb, 1999, Springer-Verlag.

[14] J. Ruf. Formal Verification of Timing Properties of a Holonic Material Transport System. *Technical Report, WSI-2000-2*, Tuebingen University, Feb. 2000.

[15] J. Ruf. RAVEN: Real-Time Analyzing and Verification Environment, *Technical Report, WSI-2000-3*, Tuebingen University, Feb. 2000.

[16] U. Schneider. Ein formales Modell und eine Klassifikation für die Fertigungssteuerung - Ein Beitrag zur Systematisierung der Fertigungssteuerung. *Dissertation,* Heinz Nixdorf Institut, 1996, HNI-Verlagsschriftenreihe. (in German)

[17] A. L. Turk, S. T. Probst, G. J. Powers. Verification of Real Time Chemical Processing Systems. In O. Maler, ed., *Hybrid and Real-Time Systems*, pp. 259-272, LNCS 1201, Springer-Verlag, 1997.

[18] E. Westkaemper, M. Hoepf, C. Schaeffer. Holonic Manufacturing Systems (HMS) - Test Case 5. In *Proc. of Holonic Manufacturing Systems*, Lake Tahoe, CA, USA, Feb. 1994.

[19] P. T. Whelan. SEMATECH's CIM Application Framework: A New Paradigm For Manufacturing Systems. In *Proc. of the Int. Conf. on Improving Manufacturing Performance in a Distributed Enterprise: Advanced Systems and Tools*, pp. 43-52, Edinburgh, 1995.