LSAT: Specification and Analysis of Product Logistics in Flexible Manufacturing Systems

Bram van der Sanden[1], Yuri Blankenstein[1], Ramon Schiffelers[2], Jeroen Voeten[3]

***Abstract*— LSAT (Logistics Specification and Analysis Tool) is a tool for rapid design-space exploration of supervisory controllers that steer the product logistics and orchestrate the behavior in flexible manufacturing systems. LSAT enables lightweight modeling of system resources, system behavior, and timing characteristics. The tool provides various visualizations to explore the controlled system behavior and analysis and optimization techniques to improve the system performance. Compared to existing approaches, LSAT provides concise modeling using languages tailored towards the application domain, with domain concepts are elements of the language. LSAT provides efficient performance analysis by exploiting the structure of the models. In this paper, we describe the rationale for developing LSAT and position it with respect to other performance modeling and analysis tools. We illustrate the benefits of LSAT with an example system.**

I. Introduction

Modern flexible manufacturing systems (FMS), such as those developed in the semiconductor and automotive industry are very complex. This complexity is often caused by the flexibility to deal with multiple product types, strict timing requirements, a large number of manufacturing steps, and a platform with many parallel shared resources. Each resource constitutes multiple peripherals, that should be orchestrated to obtain an optimal manufacturing process. FMS typically have strict productivity requirements, directly linked to profitability and cost-of-ownership per product [1]. Optimizing productivity in early system design phases is important to keep total system costs down and to guarantee efficient engineering.

The performance of an FMS strongly depends on the choices that are made by the supervisory controller and the mechanical layout. Such a controller orchestrates the resources to establish an optimal manufacturing process. To improve the system performance, it is typically not possible to rely solely on experience. Manual design-space exploration is time-consuming, and it is difficult to reason about and analyze system designs that are very different from existing solutions.

To address the performance challenges in the early design-phase of an FMS, we use a model-based performance engineering method [2]. We use precise and explicit *formal models* to capture the platform and product logistics scenarios [3]. The performance characteristics of the system can be automatically analyzed with these models. This contrasts with typical approaches that rely on experience and rules of thumb for design-space exploration and performance analysis. Simpler models, like spreadsheet models, are useful for answering basic questions about the system performance, but they are not sufficient to take into account essential behavioral details for accurate performance predictions. For the latter, methods are needed that use models with an accurate characterization of the timing behavior.

[1]ESI (TNO), Eindhoven, The Netherlands
[2]ASML, Eindhoven, The Netherlands
[3]Eindhoven University of Technology, Eindhoven, The Netherlands
E-mail: bram.vandersanden@tno.nl, yuri.blankenstein@tno.nl, ramon.schiffelers@asml.com, j.p.m.voeten@tue.nl

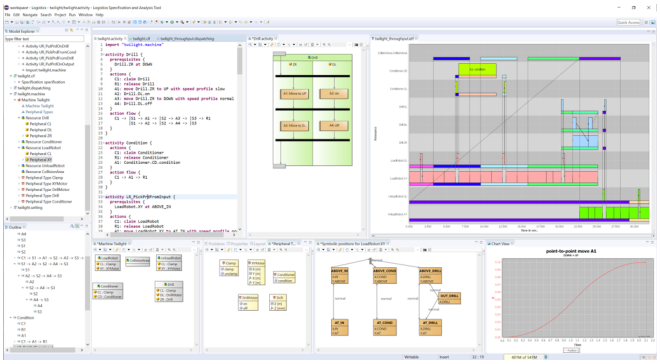

Fig. 1. A screenshot of LSAT. On top are the textual and graphical editor for activities, and a Gantt chart visualization of the system behavior. Below are three graphical editors for the system specification, and a visualization of a third-order point-to-point motion profile.

LSAT (Logistics Specification and Analysis Tool), shown in Fig. 1, provides a model-based design method with a set of domain-specific models to unambiguously describe the nominal "good weather" behavior of individual system scenarios. It is tailored towards FMS where robots move over pre-defined path segments, in contrast to systems where the movement is unrestricted, e.g., with portable robots moving freely around a factory floor. The key contribution of LSAT is that the models can capture the domain in a concise way, and the model structure can be exploited to improve scalability in the performance analysis. The models have sufficient detail to enable design-space exploration in the early-design phase of a system. Only the necessary details are modeled, to avoid a time-consuming modeling process. Because the models capture the desired system behavior in a precise and complete way, they can replace documentation and also serve as a formal contract towards a lower-level implementation. In our evaluation with industrial use cases, we found that the way of modeling matches closely the state-of-practice. During the system integration and validation phase, the implementation can be validated against the models using conformance checking.

The abstraction level of the models in LSAT is tailored

towards the early design-phase. Currently, LSAT supports modeling of individual system scenarios containing flexibility in the routing of products. For each scenario, guarantees can be given on the highest-achievable and worst-case performance by choosing different routings. For the future, we envision that LSAT will become an automated design environment, in which the LSAT specifications are exploited as configurations (i.e. designs) of a so-called platform [4] consisting of a collection of configurable components (such as physical robots or execution engines to execute activity sequences). To achieve this goal, the models should cover multiple system scenarios and automatically switch between them based on feedback from the system. This enables the modeling of exceptional cases, like failures in the system, maintenance scenarios, and product routing characteristics that are dependent on specific inputs given to the system.

In this paper we describe the reasons for developing LSAT and the design choices taken in the tool. We illustrate how the tool can be used to model and analyze supervisory controllers of FMS. The semantics of LSAT models are defined by *activity models* [5] that enable various types of performance analysis, including throughput, latency, bottleneck, and critical path analysis. LSAT has been developed in a collaboration between ESI (TNO), ASML, and Eindhoven University of Technology. The tool and documentation are available at `http://lsat.esi.nl`. Currently, LSAT is being open-sourced as an Eclipse project; see `https://projects.eclipse.org/projects/technology.lsat`.

The remainder of this paper is structured as follows. Section II positions LSAT with respect to other available tools. Section III describes the rationale of the design choices taken in LSAT, focusing on the distinction between specification and analysis models. Section IV explains the Twilight system [5] that is used in Section V to illustrate the specification modeling concepts and the types of analysis that can be performed on the models. The industrial value of LSAT is described in Section VI. Section VII concludes.

II. Related work

Using formal models for performance analysis and design-space exploration at the systems level has been advocated for a long time [6]. There are many academic and commercial tools available for these purposes. In this section we summarize related tools and position LSAT with respect to them.

In academia, there are many powerful verification and simulation tools that focus on specific types of analysis. To support a broad range of systems, in contrast to LSAT, these tools have a generic syntax, not directly tailored towards a specific application domain. Consequently, domain concepts must be encoded in these models, making them less concise. Example of these tools are POOSL [7] for performance simulation, and UPPAAL [8] and mCRL2 [9] for formal verification. Because of the generic syntax, the explicit structure typically found in FMS behavioral models can often not be exploited to increase scalability of the analysis.

One of the most well-known commercial tools for model-based design of dynamic systems (including FMS) is Simulink [10] developed by the MathWorks. Simulink contains a time-based simulation engine, and can be extended with a discrete-event simulation engine via SimEvents [11]. Control logic can be modeled using Stateflow [12]. The models in SimEvents are typically event-driven, and can be analyzed for performance characteristics such as latency and throughput. Similar to the academic tools, the manufacturing domain concepts need to be encoded in the models. In [13], the modeling approach with Simulink/SimEvents is demonstrated for evolvable production systems. Both system components like robots and grippers, and the controllers that steer the routing of products are modeled by discrete-event agent models. A case study shows how the assembly time is analyzed for a small assembly system. Automatic code generation and easily reconfiguration of local product routings are mentioned as strengths of this approach. Increased simulation time and modeling effort for larger systems are mentioned as limitations.

The activity models used in LSAT are related to activity diagrams that can be expressed in UML [14] or SysML [15]. The main use of activity diagrams in UML and SysML is in specification, and little support for performance analysis exists, whereas the formal LSAT models can be used for subsequent analysis purposes and their specific structure is exploited to do this efficiently.

MechatronicUML [16] has been created for model-driven design of mechatronic systems. Like LSAT, it has formally defined semantics including a notion of time. This enables formal analysis using UPPAAL, for example to check time-dependent safety properties. In MechatronicUML, the focus is mostly on software, whereas in LSAT, the focus is on the early design-phase where the physical layout of the system and the controller design are designed in tandem.

III. Approach: dedicated specification and analysis modeling

LSAT supports model-based design of FMS by providing dedicated specification and analysis modeling.

A. Specification modeling

LSAT provides four integrated domain-specific languages (DSLs) as shown in Fig. 2 to create *specification models* of the structure and behavior of the platform as well as the supervisory controller that orchestrates the behavior and steers the product flow. An advantage of using DSLs is that *all modeling concepts are first-class citizens of the languages*, which targets the concise modeling of the required domain concepts and nothing but those concepts. A big advantage of this approach is that the models are much easier to use, as they can intuitively be understood. This is in contrast with the typical available academic and commercial tools where domain concepts are *encoded* in another language, thereby obscuring the actual specification intent.

Platform modeling: The platform of an FMS is typically re-configurable and provides capabilities to handle products with different recipes in different ways. The platform consists of *resources*, like robots and processing stations.

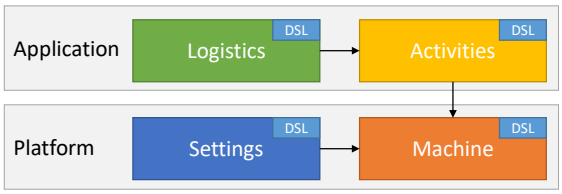

Fig. 2. DSL structure of LSAT. The arrows represent dependencies, e.g. the Logistics DSL models possible orderings of activities, and those activities are modeled with the Activities DSL.

Each resource is composed of multiple *peripherals*. Those peripherals need to work in parallel to perform a certain task. For example, a robot contains multiple motors for multiple axes to move between locations. We express those locations using *symbolic locations* and *physical locations* to decouple the logical implementation from the physical specification. Point-to-point movements are modeled in terms of the source and target location, and a *motion profile* that specifies the trajectory by a set of parameters. For example, a third-order point-to-point motion profile is described by a velocity, acceleration, and jerk parameter. Other actions in the platform can be modeled by a fixed timing value or exhibiting random variations modeled by stochastic distributions. For performance analysis in the early design-phase, this level of detail is sufficient, requiring no detailed physical models. Such models can be added in later engineering phases towards the implementation, for example using tools like Simulink [10], Modelica [17], or SimScape [18].

To model the platform, we distinguish between the *Machine model* and the *Settings model* as shown in Fig. 2. The Machine model specifies the resources, peripherals, and the actions that the peripherals can execute. The Settings model captures the physical settings of the machine, including the physical coordinates and motion profiles with their parameters. The timing of actions is described by means of distributions or constants. The timing for movements can be derived directly from the motion profiles together with the physical coordinates. Separating the settings specification from the machine specification enables design-space exploration to analyze the impact of selecting different physical settings.

Application modeling: The machine model describes all atomic *actions* that the peripherals can perform. These actions are combined in functional specification entities called *activities*. An activity describes an end-to-end deterministic system operation, such as picking up a product at a processing station and placing it on the next processing station. An activity consists of a set of actions and (acyclic) precedences among those actions. Action orderings over multiple activities are enforced through resource claims and releases. Actions of (different) activities that run on the same resource are executed sequentially. Actions on different resources are executed concurrently, taking into account the precedences specified in the activities. Within an activity, no decisions are made that alter the product flow. Such decisions are made at the supervisory machine control level, where different activity sequences represent different product flows.

LSAT is designed to conform to a scenario-based design paradigm [3], where high-level activities are defined, and possible activity orderings are defined in a *logistics* model (shown in Fig. 2). A logistics model defines the supervisory controller, which can be specified as a network of finite-state automata, or as sequences of activities that should be executed (repetitively). For case studies of logistics models for industrial systems we refer to [19], [20], [21].

B. Analysis modeling

Specification models as described in Section III-A, can be analyzed by transforming them into *analysis models*. The purpose of the analysis models is not specification, but to effectively and efficiently compute performance metrics or validate (non-)functional properties. The explicit distinction between specification and analysis has two main advantages:

- **Efficient analysis:** The required expressiveness of the analysis models can be aligned with the expressive power of the specification models. In analysis, there is always a trade-off between expressiveness and tractability of the approach [22]. The specific structures and semantics of a DSL can be exploited to enable efficient analysis.
- **Re-use of generic analysis formalisms:** Generic analysis formalisms exist for many performance questions, and are optimized to compute performance metrics effectively and efficiently for different classes of systems. With separate analysis models, all aspects from the specification language that are not important can be abstracted from. Required concepts are *encoded* in the analysis model and are not meant to be interpreted by humans.

Performance analysis in LSAT is based on max-plus linear systems theory [23]. In the transformation from specification to analysis models, motion profiles in combination with physical locations are abstracted to their timing durations. Each activity is transformed into a concise max-plus matrix [5], capturing the timing behavior induced by all action executions and their precedences inside the activity. The supervisory controller is captured by a network of max-plus automata [24]. From the matrices and automata, a max-plus state space is generated, capturing all possible system timing behavior. As shown in [5], the use of these abstractions and max-plus systems theory significantly improves the scalability of performance analysis. Scalability is further improved with a partial-order reduction [25] that exploits redundancy in the supervisory controller. Such redundancy is present when different activity sequences induce the same timing behavior and have the same functional properties. In addition, structural properties (such as confluence) of the automata can be exploited to reduce the max-plus state space [26].

LSAT supports various types of performance analysis:

- **Critical path analysis** to identify the actions that directly affect the makespan of the schedule. The critical path can be analyzed in a stochastic setting to identify how often executed actions are on the critical path given

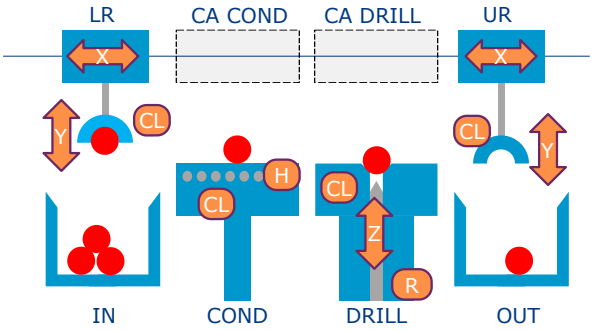

Fig. 3. Twilight manufacturing system with two robots (LR, UR) and two production stations (COND, DRILL). Figure adapted from [5].

timing probability distributions [27]. The (stochastic) critical paths are visualized using Gantt charts.

- **Throughput and makespan analysis** can be performed on the max-plus state space using maximum cycle ratio algorithms [28] for throughput and the Bellman-Ford algorithm [29], [30] for makespan. The worst-case and best-case can be analyzed to give guarantees on the minimum and maximum achievable performance. The analysis also provides the corresponding activity sequence that the supervisory controller should execute.

IV. Use Case

To illustrate the functionality of LSAT, we use the Twilight system [5], [19], shown in Fig. 3. Twilight is an imaginary manufacturing system that processes balls according to a given recipe. This system is a simplification of a lithography scanner, as modeled in [20], using similar types of resources and having similar performance optimization criteria. Twilight contains four resources. There are two robots on a rail to transport balls; a load robot (LR) and an unload robot (UR). Each robot has a homing position; LR on the left-side of the rail, and UR on the right-side of the rail. The LR is able to pick unprocessed balls from the IN position. Likewise, the UR is able to deliver processed balls to the OUT position. Next to the robots, there are two processing stations; the conditioner (COND) conditions the ball temperature, and the drill (DRILL) drills a hole in a ball. The robots have three peripherals; a clamp (CL) to pick up and hold a ball, an X-motor (X) to move along the rail, and a Y-motor (Y) to move the clamp up and down. As each robot can reach both the conditioner and the drill, there is a collision area. We partition this area into an area above the conditioner (CA COND), and the area above the drill (CA DRILL) to have a more fine-grained control of which robot is in what part of the collision area. Both the drill and conditioner have a clamp peripheral. The conditioner also has a heater (H), to heat a ball. The drill has a Z-motor (Z) to move the drill bit up and down and an R-motor (R) to rotate the drill bit.

Twilight has two high-level requirements; (1) each ball that is processed by the system should first be conditioned and then be drilled by the system, and (2) the robots should never collide with each other. The controller is constructed in such a way that these requirements are guaranteed.

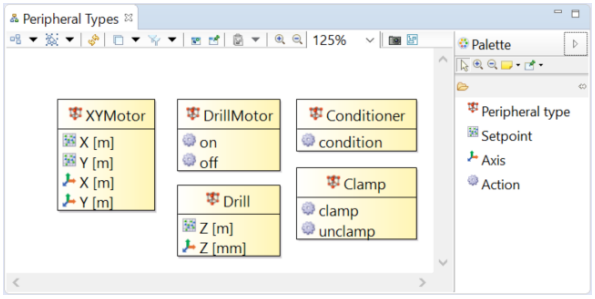

Fig. 4. Graphical editor to model the peripheral types.

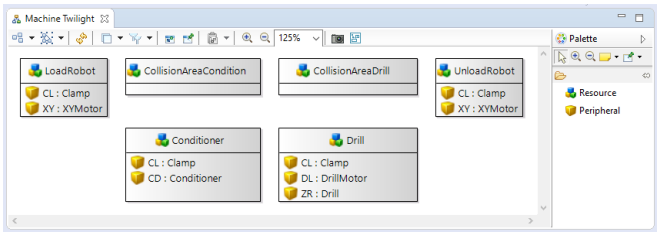

Fig. 5. Graphical editor to model the system resources.

Twilight is sufficiently small so that both the specification and the analysis results can easily be understood. In the next section, we show how to identify bottlenecks on the critical path, how to analyze the throughput, and how to improve the system design using a design-space exploration.

V. Modeling and Analysis Tool Support

In this section, we use Twilight to illustrate the modeling and analysis capabilities of LSAT. First, we model the platform and the operations. Then, we model a supervisory controller and analyze the performance of the controlled system. We show how LSAT can be used for throughput optimization, design-space exploration, and validating the implementation against the model.

A. Modeling

a) Platform modeling: We start with modeling the platform, using the Machine and Settings DSLs (Fig. 2). We use the Machine DSL to specify the available peripheral types, as shown in Fig. 4. For example, we have a clamp peripheral type, and an XYMotor peripheral type that represents a motor that can move along an X and Y-axis. The peripheral types are specified once, and can then be instantiated as part of specific resources. For instance, the XYmotor can be instantiated for the robots to model the specific locations along these axes where the robot can stop.

Fig. 5 shows the graphical resource editor to model the resources and their peripherals. The collision areas are also modeled as resources (without peripherals) that can be claimed and released, and are used to avoid the robots from colliding. Next to the graphical resource editor, there is also a textual editor. Changes in one editor are propagated to the other editor, by sharing a single internal data structure.

Next, the symbolic locations on the movement axes of a peripheral are specified using the graphical editor shown

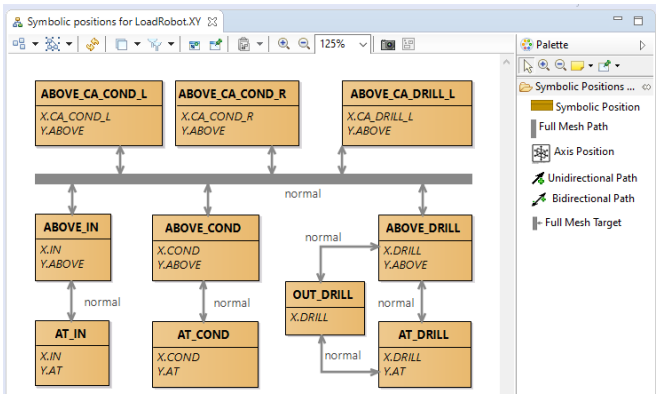

Fig. 6. Graphical editor to model the symbolic locations.

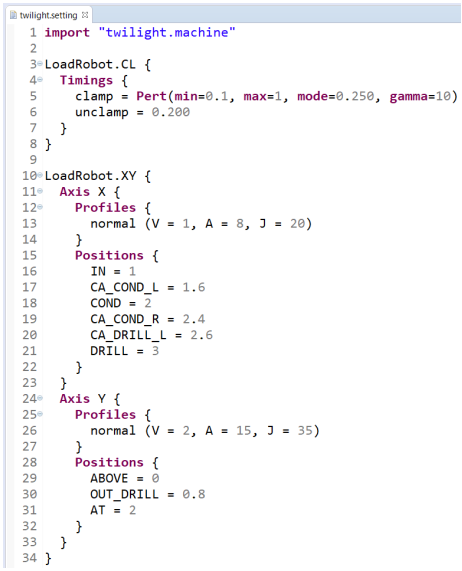

Fig. 7. Textual editor to model the physical locations, the mapping between symbolic and physical locations, and the timing of actions.

in Fig. 6. Symbolic locations of a peripheral are specified in terms of the symbolic locations per axis. For example, the location above the input buffer for the load robot, ABOVE_IN, corresponds to the IN location at the X-axis and the ABOVE location at the Y-axis. For each collision area, we model the left and right boundary points, such that the area is claimed at the time that a robot enters or leaves it. The possible movements between symbolic locations are modeled by paths. The load robot can move between any of the locations above the processing stations. This fact is modeled using a full mesh, so that not all paths between any two of these locations have to be specified individually.

With the Settings DSL, we model the physical aspects of the system. First, we describe how symbolic locations are mapped onto their physical counterparts. The mapping between symbolic and physical locations can be specified using the editor shown in Fig. 7 for the XY-motor of the load robot. Next to the locations, we also specify how movements between locations are performed. Each movement is specified using a motion profile, defined by a set of parameters. For Twilight, we use third-order point-to-point motion profiles for the movements, modeled by a velocity, acceleration, and jerk parameter (as shown on line 13 in Fig. 7). The duration of each movement is computed by LSAT from the specified parameter values and physical locations. For actions that are not concerned with a movement, the duration is specified using a constant or a stochastic distribution. LSAT supports normal, PERT, and triangular distributions, as well as a sampled distribution that is specified by a list of values. The latter is useful when modeling an existing system to get a first-order approximation of the underlying timing distribution. Line 5 in Fig. 7 shows an example, where the clamping time of the load robot clamp is specified by means of a PERT distribution.

b) Application modeling: In the platform model, we have captured all possible system behavior including the timing characteristics. Next, we model the system-level operations as activities using the Activity DSL. For Twilight, the operations include moving a ball between buffers and production stations by the robots, and the conditioning and drilling. Fig. 8 shows both the textual and graphical editor for the LRBallFromInToCond activity, that represents the operation to load a ball from the input and to bring it to the conditioner. The activity consists of a set of actions that are provided by peripherals, and dependencies between these actions. Movement actions are indicated by the *move* keyword, and specify the target location and motion profile that is used. Other actions are denoted by their explicit (hierarchical) names, such as LoadRobot.CL.clamp. Before actions of a peripheral can be executed, the corresponding resource must be claimed. We use the same resource-claiming mechanism for the collision areas, as they are normal resources (without peripherals). Before the load robot can enter the area above the conditioner, it must claim this area.

The ordering of activities is defined by a supervisory controller that steers the system to establish a correct product flow. For quick exploration of the system behavior, the supervisory controller can be specified with the Logistics DSL as a repeatable activity sequence, shown in Fig. 10. A network of automata is used for the more general case to model product routing flexibility, shown in Fig. 9. The outgoing edges in an automaton refer to the allowed activities in the given automaton location. Multiple edges represent a scheduling choice where multiple activities are allowed. Automata are synchronized on the activity names with multi-party synchronization. This means that an activity can only be executed in the current system state, if it is enabled in the current state of each of the synchronizing automata [20].

Using automata, the supervisor can be modeled directly, or alternatively one can model the uncontrolled system and the requirements using automata, and use synthesis [31], [32] to automatically compute a supervisory controller that guarantees that requirements are met in the supervised system. This functionality is provided in LSAT by transforming the model to an analysis model in CIF [33]. Fig. 9 shows part of the logistics model with three requirements, each

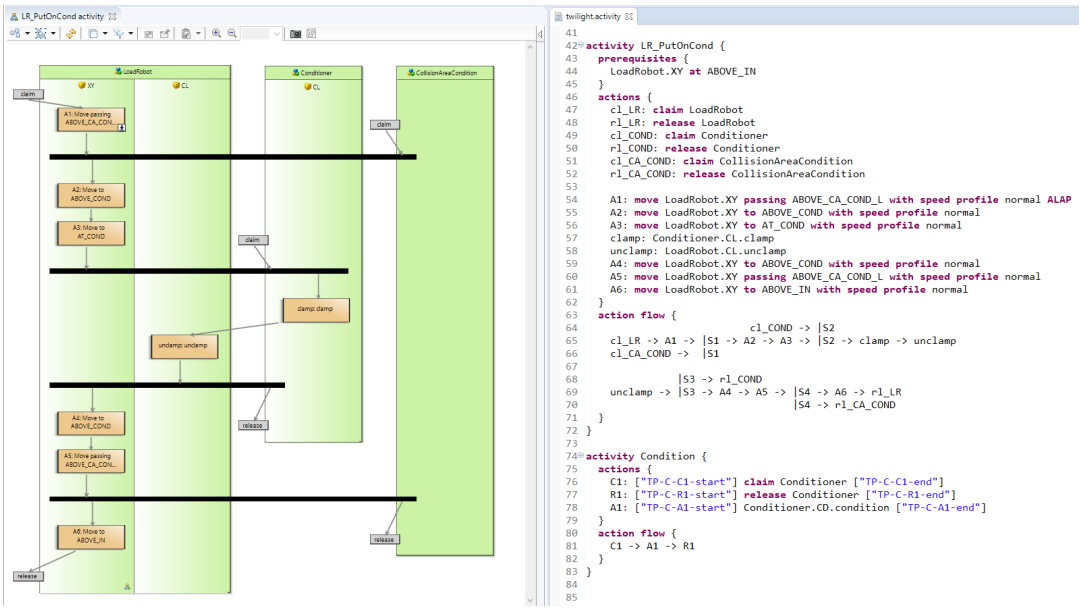

Fig. 8. Textual and graphical editor to specify activities.

```
requirement UnloadRobotMaximumOneProduct:
  location free:
    initial; marked;
    edge UR_PickFromCond, UR_PickFromDrill goto occupied;
  location occupied:
    edge UR_PutOnDrill, UR_PutOnOutput goto free;
end

requirement LoadRobotMoveBallsForward:
  location idle:
    initial; marked;
    edge LR_PickFromInput goto l1;
    edge LR_PickFromCond goto l2;
  location l1:
    edge LR_PutOnCond goto idle;
  location l2:
    edge LR_PutOnDrill goto idle;
end

requirement ConditionOnlyWhenProductPresent:
  location l0:
    initial; marked;
    edge LR_PutOnCond goto l1;
  location l1:
    edge Condition goto l2;
  location l2:
    edge UR_PickFromCond, LR_PickFromCond goto l0;
end
```

Fig. 9. Part of the logistics model capturing all possible activity sequences.

constraining the allowed activity orderings. Requirement UnloadRobotMaximumOneProduct describes that only one product can be on the unload robot at a time. A similar requirement is present for the resources LR, COND, and DRILL. Requirement LoadRobotMoveBallsForward models that the load robot always moves products forward. A similar requirement is present for the UR. Requirement ConditionOnlyWhenProductPresent models that the conditioner can only perform an operation when a product is present. A similar requirement is present for the drill.

c) Editing support and model validations: Providing both graphical and textual editors helps with understanding and developing the model. In the textual editor it is easy to make modifications, and reuse model fragments across different designs. The graphical editor on the other hand has a lower learning curve, as one does not have to learn the grammar of the textual language, and is beneficial when presenting the models to stakeholders.

To ensure a consistent model, LSAT provides the user with design-time feedback by showing a warning or error at the specific location in the model. These validations ensure that issues are identified early and do not lead to problems in the analysis models. There are syntax validations to check references to model elements like actions, peripherals, and resources. This helps the user to prevent issues caused by typographical errors. LSAT also has domain validations, to ensure that no invalid motion profiles or destinations are specified. Each activity is checked for a proper structure (formally defined in Section 2 of [5]), adhering to the resource claiming rules, and being acyclic. Analysis validations check completeness of the model, for example making sure that no physical settings are missing. The validated model can be used as input for performance analysis and optimization.

B. Performance analysis and optimization

The system behavior is analyzed by transforming the specified model to an analysis model. For performance analysis, we use a (max,+) state space that captures all possible timing behavior of the system. An optimal activity sequence is found in this state space with makespan analysis for models without cycles and throughput analysis for models with cycles.

In our Twilight logistics model, shown in Fig. 9, we have scheduling choices. Using LSAT's throughput analysis algorithm, we compute the optimal choice in every system state. The result is a repeatable activity sequence, shown in Fig. 10, where the load robot and unload robot are used alternately to move the balls from the conditioner to the drill.

The optimal sequence can be further analyzed by visualizing the corresponding Gantt chart, as shown in Fig. 11. Using stochastic critical-path analysis [27], we introduce small timing variations in the execution times of actions

```
import "platform:/resource/twilight-passing_moves/twilight.activity"

activities {
    // Throughput-optimal cycle
    LR_PickFromInput
    UR_PickFromCond
    LR_PutOnCond
    UR_PutOnDrill
    Drill
    Condition
    LR_PickFromCond
    UR_PickFromDrill
    UR_PutOnOutput
    LR_PutOnDrill
    Drill
    UR_PickFromDrill
    LR_PickFromInput
    UR_PutOnOutput
    LR_PutOnCond
    Condition
}
```

Fig. 10. Throughput-optimal supervisory controller sequence.

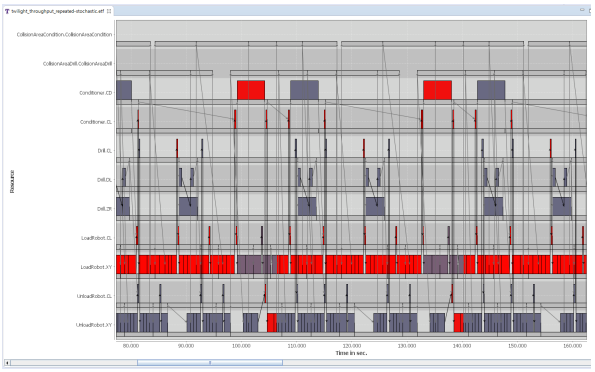

Fig. 11. Stochastic critical-path analysis on the throughput-optimal supervisory controller sequence.

based on the probability distributions in the settings file and observe how often each action is on the critical path. Actions that are often on the critical path are colored dark red. From the Gantt chart, we can observe that the load robot and the conditioner are both bottlenecks, as most actions on the critical path are executed on these resources. Using LSAT, we can start a design-space exploration to analyze the impact on speeding up either the conditioner or the load robot. We updated the model to double the maximum velocity in the X-direction of the XY-motor of the load robot. Analysis on the updated model shows that with this modification, a makespan improvement of 9.1% can be achieved.

C. Connecting the models with the implementation

a) Conformance checking: Models expressed in LSAT can be used as a formal specification of the behavior that the system should exhibit. The models can be used to validate the lower-level implementation by means of linking the execution log to the LSAT model. This connection is made by extending the activity models with (optional) trace points that identify the start and end points of actions. Fig. 8 shows an example for the Condition activity in the textual specification, where the start and end of each action has a unique trace point. For instance, the start and end of the condition action have trace points TP-C-R1-start and TP-C-R1-end respectively. The conformance check takes the execution log of the system and verifies whether the ordering of the trace points in the log matches the structure of the activities. If not, a visualization is generated that shows the trace points that cannot be matched.

b) Controller implementation: The models specified in LSAT can be used as a starting point for generating the implementation. From the activity models, a proprietary plugin has been developed to generate activity templates following a company-proprietary code architecture. Using the conformance checking, the executions of the implementation can be validated against the model.

LSAT has been used as a starting point to generate a feed-forward controller [19] for the xCPS [34], [35] platform, a machine that mimics a production line capable of assembling and disassembling cylindrical objects. The platform controller is built as an event-driven program with callbacks. Actions in the activity models are mapped onto available xCPS API functions, that control the system peripherals. The controller computes the action-level schedule, given the activity sequence, and ensures that the corresponding actions are executed in the specified order, taking into account the resource claims and releases. Based on the callbacks, the controller knows when actions are finished, and dispatches succeeding actions to the peripherals.

VI. Industrial Use Cases

LSAT is used at ASML to create fully calibrated models of the wafer handling subsystems of a wafer scanner. This subsystem is responsible for transporting wafers in and out of the scanner, and performs preprocessing steps before the wafer is being exposed on the wafer stage subsystem. A supervisory controller for such a subsystem is described in [20]. Design-space exploration is done by ASML with LSAT to evaluate the impact of modifications on the mechanical platform architecture (e.g. different motion settings or alternative resource layouts) and different supervisory controllers on the system performance. The LSAT models have also been transformed to executable activities that are executed on the system test platform, to validate the observed Gantt charts with the specified Gantt charts, facilitating model calibration.

LSAT is also used as a design-by-contract collaboration of ASML and VDL-ETG for formal specification of the operations in the nominal product flow. Here, LSAT replaces traditional documentation, and serves as a blueprint for the nominal logistics behavior, which VDL-ETG uses to generate UML activity diagrams and code based on the specified activities.

At Nexperia Equipment & Automation, LSAT is used to model the machine cycle of die bonding machines, used to attach dies to a substrate or package. The LSAT models, which include domain-specific motion profile calculators, are (semi-automatically) calibrated with settings and action timings from existing systems. The calibrated models are used for critical-path analysis, budget analysis, and productivity predictions.

At Eindhoven University of Technology, LSAT is used as a research and education vehicle, in relation to the xCPS platform [34], [35].

VII. Conclusion

In this paper, we have introduced LSAT for rapid design-space exploration of FMSs in the early design-phase, focusing on the platform and product logistics. Compared to existing approaches, LSAT provides a modeling environment where the platform and product logistics can be modeled concisely using a compact set of domain concepts, without the need of encoding domain concepts in a generic language. Because of this, models are easier to understand and maintain, and many types of design-time feedback can be given to the user, leading to less errors and a more efficient design process. LSAT provides efficient performance analysis by exploiting the structure of the models.

The current approach has some limitations, that should be addressed in future work. LSAT does not yet support the modeling and analysis of supervisory controllers that can automatically switch between system scenario based on system feedback. For example, to model and analyze a control choice to start a failure scenario in case that an operation fails. In LSAT it is currently also not possible to model computation tasks that are multiplexed on computational resources. As such tasks get more compute intensive (e.g., when dealing with large volumes of data), they might impact the critical path of the product flow and should be taken into account.

Acknowledgment

This research is carried out as part of the Concerto and Maestro projects, under the responsibility of ESI (TNO) with ASML as the carrying industrial partner. Research leading to these results has received funding from the Dutch NWO-TTW, carried out as part of the Robust Cyber-Physical Systems (RCPS) program, project number 12694, and from the EU ECSEL Joint Undertaking under grant agreement no 826452 (project Arrowhead Tools). We would like to thank Twan Basten, Marc Geilen, Michel Reniers, and Johan Jacobs for their contributions to LSAT.

References

[1] D. W. Jimenez, "Cost of ownership and overall equipment efficiency: a photovoltaics perspective," *Photovoltaics International Journal*, pp. 16–22, 2009.

[2] M. Hendriks, J. Verriet *et al.*, "Performance engineering for industrial embedded data-processing systems," in *Product-Focused Software Process Improvement*, ser. LNCS, vol. 9459. Springer, 2015, pp. 399–414.

[3] T. Basten, J. Bastos *et al.*, *Scenarios in the design of flexible manufacturing systems*. Springer, 2019.

[4] A. Vincentelli, "Quo vadis, SLD? reasoning about the trends and challenges of system level design," *Proc. of the IEEE*, vol. 95, pp. 467 – 506, 04 2007.

[5] B. van der Sanden, J. Bastos *et al.*, "Compositional specification of functionality and timing of manufacturing systems," in *Proc. of the Forum on Spec. and Design Languages*. IEEE, 2016, pp. 1–8.

[6] J. Fowler and O. Rose, "Grand challenges in modeling and simulation of complex manufacturing systems," *Simulation*, vol. 80, no. 9, pp. 469–476, Sep. 2004.

[7] J. Voeten, *POOSL: An object-oriented specification language for the analysis and design of hardware/software systems*. Eindhoven University of Technology, Faculty of Electrical Engineering, 1995.

[8] J. Bengtsson, K. Larsen *et al.*, "UPPAAL — a tool suite for automatic verification of real-time systems," in *Int. hybrid systems workshop*, ser. LNCS, vol. 1066. Springer, 1995, pp. 232–243.

[9] O. Bunte, J. F. Groote *et al.*, "The mCRL2 toolset for analysing concurrent systems," in *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, vol. 11428. Springer, 2019, pp. 21–39.

[10] J. B. Dabney and T. L. Harman, *Mastering Simulink*. Pearson, 2004.

[11] M. I. Clune, P. J. Mosterman, and C. G. Cassandras, "Discrete event and hybrid system simulation with SimEvents," in *Proc. of the 8th Int. Workshop on Discrete Event Systems*, 2006, pp. 386–387.

[12] W. Li, R. Mani, and P. J. Mosterman, "Extensible discrete-event simulation framework in SimEvents," in *2016 Winter Simulation Conf.* IEEE, 2016, pp. 943–954.

[13] A. Rahatulain, T. N. Qureshi, and M. Onori, "Modeling and simulation of evolvable production systems using Simulink/SimEvents," in *Conf. of the IEEE Industrial Electronics Society*. IEEE, 2014, pp. 2591–2596.

[14] S. Cook, C. Bock *et al.*, "Unified modeling language (UML) version 2.5.1," Object Management Group (OMG), Standard, Dec. 2017.

[15] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[16] W. Schäfer and H. Wehrheim, *Model-Driven Development with Mechatronic UML*. Springer, 2010, pp. 533–554.

[17] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[18] S. Miller and J. Wendlandt, "Real-time simulation of physical systems using Simscape," *Real-Time Simulation Technologies: Principles, Methodologies, and Applications*, 08 2012.

[19] B. van der Sanden, "Performance analysis and optimization of supervisory controllers," Ph.D. dissertation, Eindhoven University of Technology, 2018.

[20] B. van der Sanden, M. Reniers *et al.*, "Modular model-based supervisory controller design for wafer logistics in lithography machines," in *18th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems*. IEEE Computer Society, 2015, pp. 416–425.

[21] J. Bastos, "Modular specification and design exploration for flexible manufacturing systems," Ph.D. dissertation, Eindhoven University of Technology, 2018.

[22] H. J. Levesque and R. J. Brachman, "Expressiveness and tractability in knowledge representation and reasoning 1," *Computational intelligence*, vol. 3, no. 1, pp. 78–93, 1987.

[23] F. Baccelli, G. Cohen *et al.*, *Synchronization and Linearity*. Wiley, 1992.

[24] S. Gaubert, "Performance evaluation of (max,+) automata," *IEEE Trans. on Automatic Control*, vol. 40, no. 12, Dec 1995.

[25] B. van der Sanden, M. Geilen *et al.*, "Partial-order reduction for supervisory controller synthesis," *To appear in IEEE Trans. on Automatic Control*.

[26] J. Bastos, J. Voeten *et al.*, "Taming the state-space explosion in the makespan optimization of flexible manufactoring systems," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 2, 2021.

[27] J. Bastos, B. van der Sanden *et al.*, "Identifying bottlenecks in manufacturing systems using stochastic criticality analysis," in *Proc. of the Forum on Specification and Design Languages*. IEEE, 12 2017.

[28] A. Dasdan, "Experimental analysis of the fastest optimum cycle ratio and mean algorithms," *ACM-TODAES*, vol. 9, no. 4, pp. 385–418, 2004.

[29] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[30] L. R. Ford, "Report P-923, Network Flow Theory," The Rand Corporation, Tech. Rep., 1956.

[31] P. J. G. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.

[32] ——, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[33] D. van Beek, W. Fokkink *et al.*, "CIF 3: Model-based engineering of supervisory controllers," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS. Springer, 2014, vol. 8413, pp. 575–580.

[34] S. Adyanthaya, H. Alizadeh Ara *et al.*, "xCPS: A tool to explore cyber physical systems," in *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*. ACM, 2015, pp. 3:1–3:8.

[35] ——, "xCPS: a tool to explore cyber physical systems," *SIGBED Review*, vol. 14, no. 1, pp. 81–95, 2016.